



UNIVERSIDADE DO VALE DO TAQUARI – UNIVATES
CURSO DE ENGENHARIA DA COMPUTAÇÃO

**ANÁLISE DE DESEMPENHO ENTRE MÁQUINAS VIRTUAIS
E CONTAINERS PARA APLICAÇÕES WEB**

Gerson Fell

Lajeado, novembro de 2018.

Gerson Fell

ANÁLISE DE DESEMPENHO ENTRE MÁQUINAS VIRTUAIS E CONTAINERS PARA APLICAÇÕES WEB

Trabalho de conclusão de curso apresentado na disciplina de Trabalho de Conclusão de Curso II, do Curso de Engenharia da Computação, da Universidade do Vale do Taquari - Univates, como parte da exigência para a obtenção do título de Bacharel em Engenharia da Computação.

Orientador: Prof. Me. Luis Antônio Schneiders

Lajeado, novembro de 2018.

Dedico este trabalho a minha mãe, Alzira Petronila Klafke, que me apoiou em todos os momentos do bacharelado.

AGRADECIMENTOS

A todos os professores que durante toda graduação, colaboraram com ideias me incentivando a fazer o melhor possível. Em especial ao orientador Luis Antônio Schneiders por ter acreditado em mim e na minha ideia, colaborado no desenvolvimento deste trabalho.

A minha família e todos amigos que me incentivaram e me encorajaram durante todo o período do desenvolvimento de trabalho.

A todos os colegas pela troca de conhecimento e pelos momentos compartilhados durante toda a formação.

RESUMO

Este trabalho teve como objetivo realizar uma análise de desempenho entre dois cenários, aplicando diferentes conceitos de virtualização, com a finalidade específica de hospedagem de aplicações Web. Para tanto, foi configurado um ambiente de teste com o objetivo de explorar o desempenho de implantações de máquinas virtuais tradicionais, e assim, compará-las com o uso de *containers*. Será utilizado um conjunto de cargas de trabalho para ajudar no aumento do uso da CPU, memória, armazenamento e recursos de rede e desta forma foi possível avaliar a solução com os melhores resultados no sentido de melhor aproveitamento dos recursos computacionais. Demonstrou-se a melhor solução para virtualização de aplicativos que utiliza os navegadores Web como forma de acesso.

Palavras-chave: Virtualização. *Containers*. Aplicativos Web.

ABSTRACT

This work aims to perform a performance analysis between two scenarios, applying different concepts of virtualization, for the specific purpose of hosting Web applications. To do so, a test environment will be set up to explore the performance of machine deployments and compare them with the use of containers. A set of workloads will be used to help increase the use of CPU, memory, storage and network resources and this way it will be possible to evaluate the solution with the best results in order to better use of the computational resources. It is hoped to demonstrate the best solution for application virtualization that uses Web browsers as a form of access.

Keywords: Virtualization. Containers. Web Applications.

LISTA DE FIGURAS

Figura 1 - Definição da virtualização de sistemas operacionais.	19
Figura 2 - Virtualização com hypervisor.	20
Figura 3 - Exemplo de máquina virtual.	21
Figura 4 - Máquina Virtual de aplicação.	22
Figura 5 - Máquina virtual de sistemas.	23
Figura 6 - Modelo de virtualização tipo 1 com hypervisor monolítico.	24
Figura 7 - Modelo de virtualização tipo 1 com hypervisor microkernel.	25
Figura 8 - Modelo de virtualização tipo 2.	26
Figura 9 - Hierarquia na arquitetura do processador x86.	27
Figura 10 - Virtualização total na arquitetura x86.	28
Figura 11 - Virtualização total ou completa.	30
Figura 12 - Paravirtualização.	31
Figura 13 - Hierarquia do processador x86 na paravirtualização.	32
Figura 14 - Virtualização Assistida por Hardware na Arquitetura x86.	33
Figura 15 - Comunicação entre a rede física e máquinas virtuais.	35
Figura 16 - Servidores de hypervisor em um cluster.	36
Figura 17 - Conceito de infraestrutura de virtualização de desktops.	37
Figura 18 - Utilização de Thin Client no conceito de virtualização de desktop.	39
Figura 19 - Virtualização de aplicações.	40
Figura 20 - Arquitetura de um <i>storage</i>	41
Figura 21 - Técnicas de entrada/saída.	41
Figura 22 - Comparação da estrutura máquinas virtuais típicas e containers.	43
Figura 23 - Cluster interligado pela rede ethernet.	44
Figura 24 - Testes de leitura e gravação em disco.	50
Figura 25 - Testes de rede.	51
Figura 25 - Estrutura física dos servidores utilizado nos testes.	53
Figura 26 - Estrutura lógica do cenário 1.	54
Figura 27 - Lista de comando para instalação do Docker.	54
Figura 28 - Lista de comando para instalação do Apache e MYSQL.	55
Figura 29 - Teste de acesso externo no container Apache do Docker SRV001.	56
Figura 30 - Configuração de usuário para acesso externo MYSQL.	56
Figura 31 - Teste de acesso externo no container MYSQL do Docker SRV001.	57
Figura 32 - Estrutura lógica do cenário 2.	57
Figura 33 - Tutorial para instalação do ESXI 6.0.	58
Figura 34 - Configuração da máquina virtual SRV002.	58

Figura 35 - Comando para instalação do Apache no SRV002.	59
Figura 36 - Teste de acesso externo no Apache da máquina virtual SRV002.	60
Figura 37 - Comando para instalação do MYSQL no SRV002.	60
Figura 38 - Teste de acesso externo no MYSQL máquina virtual SRV002.	61
Figura 39 – Cenário de configuração do software Zabbix.	61
Figura 40 – Cenário de configuração do software Zabbix.	62
Figura 41 – Consumo da CPU dos servidores com uso de <i>templates</i>	63
Figura 42 – Página de administração do Grafana utilizando o Zabbix.	64
Figura 43 – Compartilhamento de gráficos utilizando o Grafana.	65
Figura 44 – Instalação do servidor de página Tomcat 7.	66
Figura 45 – Diagrama da aplicação central.	66
Figura 46 – Código de inserção e seleção na base de dados.	67
Figura 47 – Código simulando a criação múltiplos usuários.	68
Figura 48 – Aplicação mestre efetuando testes de inserção e seleção.	69
Figura 49 – Recurso de análise de logs do teste de inserção e seleção.	69
Figura 50 – Execução do Apache Bench no servidor SRV003.	70
Figura 51 – Função para executar o Apache Bench na aplicação mestre.	71
Figura 52 – Aplicação mestre configurada para efetuar teste de requisição.	71
Figura 53 – Recurso de análise de logs do teste de requisição.	72
Figura 54 - Parâmetros utilizados no teste com carga do MYSQL.	77
Figura 55 - Parâmetros utilizados no teste com carga do Apache.	80

LISTA DE GRÁFICO

Gráfico 1 – Consumo de memória sem carga.	75
Gráfico 2 – Consumo de CPU sem carga.	75
Gráfico 3 – I/O do disco rígido sem carga.	76
Gráfico 4 – Entrada e saída de rede sem carga.	76
Gráfico 5 – Consumo de memória no teste com carga do MYSQL.	78
Gráfico 6 – Consumo de CPU no teste com carga do MYSQL.	78
Gráfico 7 – I/O do disco rígido no teste com carga do MYSQL.	79
Gráfico 8 – Entrada e saída de rede no teste com carga do MYSQL.	79
Gráfico 9 – Consumo de memória no teste com carga do Apache.	81
Gráfico 10 – Consumo de CPU no teste com carga do Apache.	81
Gráfico 11 – I/O do disco rígido no teste com carga do Apache.	82
Gráfico 12 – Entrada e saída de rede no teste com carga do Apache.	82
Gráfico 13 – Consumo de memória após execução dos testes.	83
Gráfico 14 – Consumo de CPU após execução dos testes.	83

LISTA DE TABELAS

Tabela 1 - Comparação entre os modelos de virtualização.	34
---	----

LISTA DE QUADROS

Quadro 1 - Configurações gerais dos softwares utilizados.	73
--	----

LISTA DE ABREVIATURAS E SIGLAS

AMD	Advanced Micro Devices
CPL	Current Privilege Level - Nível de privilégio atual
CPU	Central Process Unit - Unidade Central de Processamento
IBM	International Business Machines
ICA	Independent Computing Architecture - Arquitetura de computação independente
I/O	Input/Output - Entrada/Saída
INTEL	Integrated Electronics
iOS	iPhone Operational System - Sistema Operacional para iPhone -
JVM	Java Virtual Machine - Máquina Virtual de Java
KVM	Kernel-based Virtual Machine - Máquina Virtual baseada em Núcleo
RAM	Random Access Memory - Memória de Acesso Aleatório
SAN	Storage Area Network - Rede de Área de Armazenamento
NAS	Network Attached Storage - Armazenamento Acoplado de Rede
PVM	Process Virtual Machine - Processador da Máquina Virtual
QEMU	Short for Quick Emulator - Emulador de software livre
TI	Tecnologia da Informação
VDI	Virtualization Desktop Infrastructure - Infraestrutura da área de trabalho de virtualização

- VM** Virtual Machine - Máquina Virtual
- VMM** Virtual Machine Monitor - Monitor de Máquina Virtual
- VNC** Virtual Network Computing - Computação de rede virtual

SUMÁRIO

1 INTRODUÇÃO	15
1.1 Tema	16
1.2 Objetivos	16
1.2.1 Objetivo geral	16
1.2.2 Objetivos específicos	16
1.3 Motivação	17
1.4 Organização do trabalho	17
2 REFERENCIAL TEÓRICO	18
2.1 Conceito de virtualização	18
2.2 Máquinas virtuais	20
2.3 Modelos de máquinas virtuais	22
2.4 Tipos de <i>hypervisor</i>	24
2.4.1 Tipo 1	24
2.4.2 Tipo 2	25
2.5 Modelos de virtualização	26
2.5.1 Virtualização completa ou total	27
2.5.2 Paravirtualização	30
2.5.3 Virtualização assistida por hardware	32
2.5.4 Comparação entre os tipos de virtualização	34
2.6 Outros conceitos e soluções de virtualização	34
2.6.1 Virtualização de rede	35
2.6.2 Virtualização de desktop	37
2.6.3 Virtualização de aplicação	39
2.6.4 Virtualização de <i>storage</i>	40
2.6.5 Containerização	42
2.7 Cluster	44
3 METODOLOGIA	46
3.1 Método de pesquisa e trabalho	46
3.1.1 Quanto aos objetivos	47
3.2 Procedimentos de pesquisa	47
3.2.1 Referencial teórico	47
3.2.2 Coleta de dados	47
3.2.3 Análise de dados	47
4 TRABALHOS RELACIONADOS	49
4.1 Comparação de desempenho de máquinas virtuais e containers	49

4.1.2 Testes executados	50
5 DESENVOLVIMENTO	52
5.1 Implementação dos cenários de testes	52
5.1.1 Cenário 1	53
5.1.2 Cenário 2	57
5.2 Coleta de dados	61
5.3 Aplicação mestre	65
5.3.1 Teste MYSQL	67
5.3.2 Teste Apache	69
5.4 Configurações gerais	72
6 TESTES E ANÁLISES	74
6.1 <i>Teste sem carga</i>	74
6.2 <i>Testes com carga do MYSQL</i>	77
6.3 <i>Testes com carga do Apache</i>	80
6.4 <i>Observações gerais sobre os testes</i>	83
7 CONSIDERAÇÕES FINAIS	84
REFERÊNCIAS	86

1 INTRODUÇÃO

Na atualidade os computadores, redes e sistemas de processamento vem conquistando um espaço de destaque nas operações das empresas. Os computadores estão se tornando cada vez menores, com uma maior capacidade de processamento, armazenamento e com melhores características de mobilidade e segurança.

Com a constante evolução dos computadores, há um grande aumento no volume de dados das empresas, e a utilização de aplicações e de serviços da Tecnologia da Informação (TI) se tornou imprescindível no dia a dia destas empresas. A consequência desta evolução é o aumento da procura por bases de dados mais velozes e confiáveis e um dos elementos fundamentais neste ambiente são os Data Center.

Segundo Golden e Scheffy (2008), muitos *data centers* atualmente possuem servidores que estão utilizando apenas 10 ou 15% da sua capacidade total de processamento, memória e disco. Em outras palavras podemos dizer que mais de 80% da sua capacidade está ociosa ou seja em desuso.

Contudo, ainda que um servidor esteja utilizando apenas 10% dos seus recursos computacionais e consumindo energia elétrica, gerando basicamente os mesmos custos operacionais que os de um servidor que está utilizando 100% de dos seus recursos computacionais. Segundo Siqueira (2007), aplicando o conceito de virtualização dentro dos data centers o problema com ocupação de espaços indevidos e desperdícios de recursos computacionais é reduzido, otimizando os recursos.

Pode ser citado como a principal vantagem da virtualização a redução de custos, pois apenas com um servidor físico é possível montar uma estrutura com vários servidores virtualizados, dispensando a compra de equipamentos desnecessários. Operacionalmente podemos dizer que também há várias vantagens, pois, a administração se torna totalmente centralizada e o tempo total de manutenção de um equipamento virtualizado é menor quando comparado a um equipamento comum (SIQUEIRA, 2007).

Outra vantagem que deve ser considerada é a utilização recursos não convencionais, como a migração de serviços tempo real, que permite que um sistema virtualizado altere seu *host* hospedeiro sem que haja alguma parada no fornecimento dos serviços deste sistema, recurso imprescindível para ambientes críticos com o requerimento de alta disponibilidade.

1.1 Tema

Máquinas virtuais e *containers* para aplicação Web.

1.2 Objetivos

Nas próximas seções serão apresentados os objetivos geral e específicos do estudo.

1.2.1 Objetivo geral

Realizar uma análise comparativa de dois cenários aplicando diferentes conceitos de virtualização, considerando os modelos de virtualização típicos e o de containerização, avaliando o processamento, consumo de memória, IO e tráfego de rede.

1.2.2 Objetivos específicos

Para comprovar uma diferença no consumo de recursos computacional entre os modelos de virtualização típica e containerização, foi analisado o consumo dos seguintes itens:

- a. Processador;

- b. Memória RAM;
- c. Rede;
- d. Disco Rígido.

1.3 Motivação

Com a crescente evolução da internet, a tendência mundial é que todos os serviços sejam migrados gradativamente para aplicações que possam ser acessadas através de navegadores Web, tanto para soluções de Internet quanto de intranet. A linguagem de programação utilizada na construção de aplicativos baseados em Web é indiferente para o usuário final. Isto possibilita a independência de plataformas e de sistemas operacionais, outra grande vantagem é a facilidade de implantação.

Considerando o aumento na demanda em aplicações Web a motivação foi pesquisar tecnologias para utilizar os recursos computacionais das empresas de uma forma mais inteligente “mais com menos” utilizando o conceito de computação em nuvem mais especificamente a Plataforma como Serviço.

Este estudo poderá contribuir como um todo, para as empresas em geral. Estas poderão usufruir deste estudo prático como tomada de decisão para implantar no seu *data center* o conceito mais eficiente de cluster com base na utilização de aplicações Web.

1.4 Organização do trabalho

O presente trabalho está organizado em capítulos. Após a introdução é apresentado o referencial teórico, que contém toda fundamentação de base sobre o tema deste trabalho. No capítulo 3 foi abordado a metodologia utilizada para o desenvolvimento deste trabalho. Já no capítulo 4 é relatado um trabalho relacionado que foi utilizado para verificar a viabilidade técnica do objetivo proposto. No capítulo seguinte será apresentado o desenvolvimento do presente trabalho. Dando sequência o capítulo 6 apresentara os testes efetuados e também será efetuado uma análise dos testes propostos. Por fim o capítulo 7 será apresentado as considerações finais do presente trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo, além das questões gerais de virtualização, serão apresentadas as teorias dos conceitos de virtualização. Também será efetuada uma abordagem sobre o conceito de clusters.

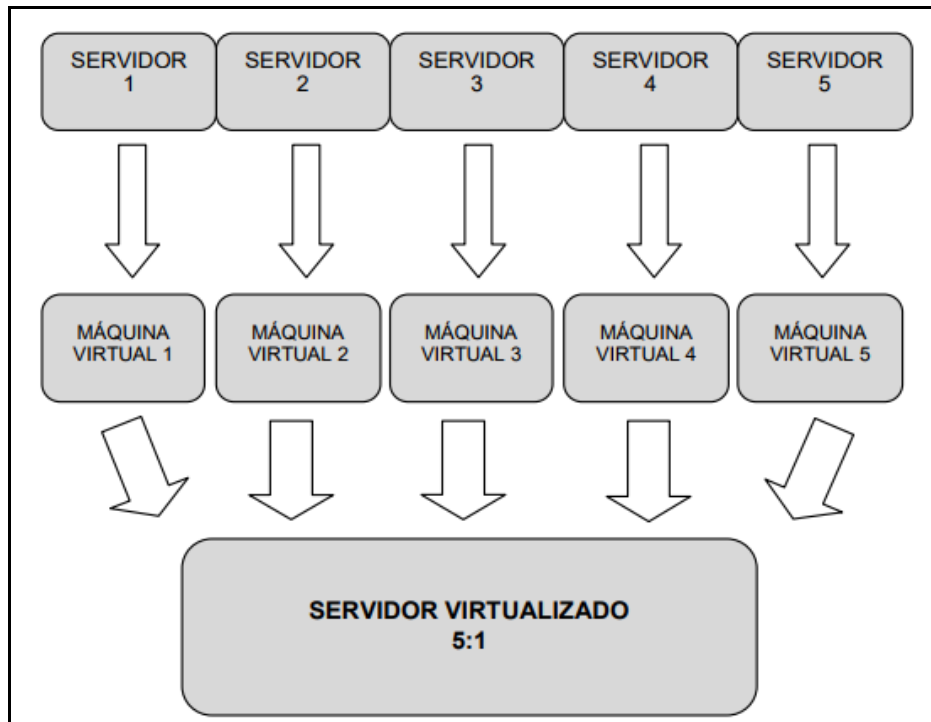
2.1 Conceito de virtualização

O conceito de virtualização foi desenvolvido pela IBM nos anos 60. O foco principal era executar várias máquinas virtuais em apenas um único *hardware*, reduzindo assim, os custos com equipamentos os quais eram muito altos na época (MATTOS, 2008).

Em 1972 o cientista americano Robert P. Goldberg apresentou a teoria da arquitetura para os sistemas computacionais virtuais em sua dissertação (GOLDBERG, 1972). No mesmo ano a IBM lançou o *mainframe* IBM S/360, desenhado para o sistema de virtualização, preparado para executar ao mesmo tempo inúmeros sistemas operacionais sob a supervisão de um *software* de controle chamado *hypervisor* (MATTOS, 2008).

Segundo Veras (2010), a virtualização pode ser conceituada de duas maneiras: virtualização de sistemas e virtualização de aplicações. A virtualização de sistemas equivale-se a instalar em um único *hardware* inúmeros sistemas operacionais, que assim possibilita a execução simultaneamente destes sistemas operacionais. Um exemplo deste conceito está na Figura 1 em que cinco servidores foram virtualizados em um único servidor físico.

Figura 1 - Definição da virtualização de sistemas operacionais.

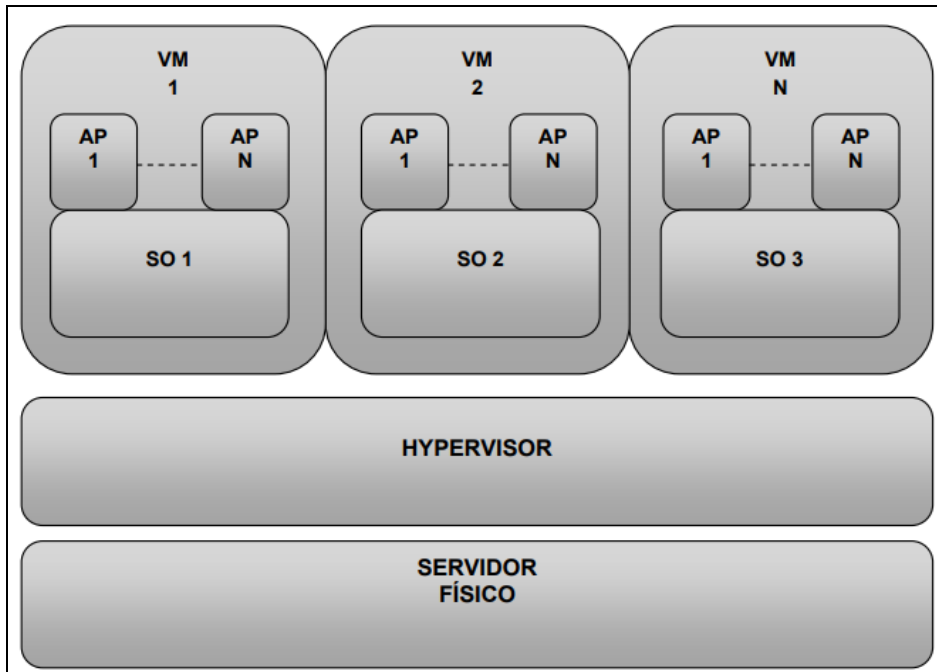


Fonte: Veras (2011, p. 87).

O conceito de virtualização de aplicações segundo Veras (2010), é uma camada de abstração entre o *software* e o *hardware* que monitora o acesso direto do *software* aos recursos computacionais do *hardware* físico. Este conceito possibilita que as camadas de *software* (sistemas operacionais e aplicações) sejam totalmente isoladas da camada do *hardware* físico, sendo assim realizada por um *software* (VERAS, 2010).

A camada de virtualização provê para o sistema operacional convidado um conjunto de instruções de máquina, como memória, processador e os demais elementos que uma máquina virtual necessite para o seu funcionamento. A camada de virtualização mais utilizada é o *hypervisor* ou também conhecida como Monitor de Máquina Virtual (Virtual Machine Monitor – VMM) (GOLDBERG, 1972). O *hypervisor* é um *software* que possibilita a hospedagem das máquinas virtuais gerenciando os recursos utilizados por estas máquinas visitantes sobre a máquina física (FIGURA 2).

Figura 2 - Virtualização com hypervisor.



Fonte: Veras (2011, p. 87).

Segundo Machado (1997), a virtualização é a simulação de um *hardware/software* que roda sobre outro *software*. Este conceito de ambiente simulado é chamado de máquina virtual (VM - *Virtual Machine*).

Desde os anos 70 foram criadas diversas definições para o que seria a virtualização. Segundo Machado (2008), a forma mais fácil de explicar este conceito, seria o fato de haver várias máquinas com sistemas operacionais distintos sendo executadas no interior de outra máquina, assim fazendo com que os recursos computacionais desta máquina real tenham um nível de otimização muito superior quando comparado a um ambiente comum sem as técnicas de virtualização.

Sendo assim, como foi citado acima, a virtualização não se trata de uma tecnologia recente. Esta tecnologia tem se mostrado mais eficiente atualmente tendo em vista a sua constante evolução nos últimos anos.

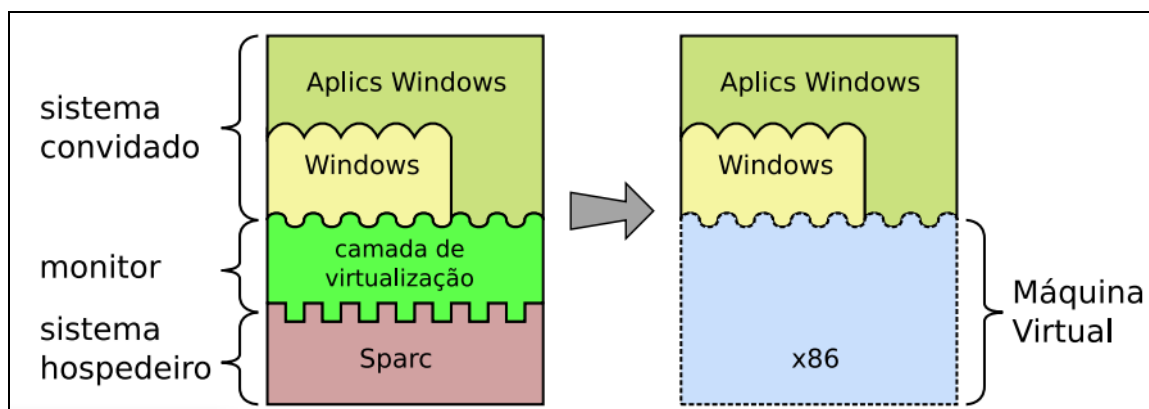
2.2 Máquinas virtuais

Segundo Maziero (2017), para que seja possível executar bibliotecas e programas sobre um *hardware* diferente, é necessário que o sistema operacional tenha sido compilado para a plataforma desejada. Atualmente nos sistemas as

interfaces de baixo nível são pouco flexíveis e geralmente não é possível criar novas chamadas de sistema ou novas instruções de processador. Por este motivo o autor afirma que um sistema operacional só pode ser executado por um *hardware* para o qual foi projetado.

De acordo com Barros (2016), para conseguir resolver os problemas de compatibilidade entre todos os componentes de um sistema operacional através de técnicas de virtualização é necessário que seja implantada uma camada de *software*. Essa camada vai garantir assim, o acoplamento entre as interfaces diferentes, de tal forma que um programa projetado para um *hardware* específico possa ser executado normalmente em outros *hardwares* diferentes (FIGURA 3).

Figura 3 - Exemplo de máquina virtual.



Fonte: Do autor, adaptado de Smith e Nair (2004).

Na Figura 3 foi apresentado um sistema operacional Windows que atualmente é projetado somente para os *hardwares* Intel e Amd e está sendo executado normalmente sobre uma camada de virtualização que está alocada sobre um *hardware spark*.

Segundo Veras (2011), um ambiente com máquinas virtuais baseia-se em três componentes básicos:

- O sistema hospedeiro ou sistema real tem acesso aos recursos reais de *hardware*;
- O sistema virtual ou sistema convidado, que é executado sobre um sistema hospedeiro;

- A camada de virtualização, conceituada de *hipervisor* ou monitor de virtualização, que é responsável por construir as interfaces virtuais a partir de uma interface real.

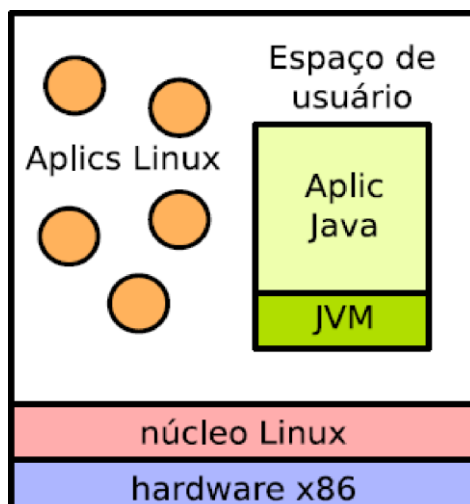
2.3 Modelos de máquinas virtuais

Os sistemas de virtualização gradativamente estão conquistando espaço, em qualquer tamanho de empresa. Isso pode estar ocorrendo em virtude da fácil assimilação das vantagens associadas a essa plataforma (VERAS, 2010).

Os modelos de máquinas virtuais diferenciam-se dependendo de qual é o objetivo da virtualização, isto é, o que se espera obter com a adoção de um determinado sistema de virtualização. As máquinas virtuais podem ser conceituadas de duas formas: virtualização de aplicações ou a virtualização de sistemas (LAUREANO, 2006).

Máquinas virtuais de aplicações vem do inglês *Process Virtual Machine* que se resume em um ambiente de máquinas virtuais montado para suportar apenas um único processo ou uma determinada aplicação convidada. Pode ser apresentado como um exemplo a máquina virtual do Java que é uma solução de virtualização preparada para executar os *bytecodes* em qualquer plataforma, independentemente da plataforma que o código fonte da aplicação foi compilado. A Figura 4 ajuda a ilustrar um exemplo comum de máquina virtual de aplicação.

Figura 4 - Máquina Virtual de aplicação.

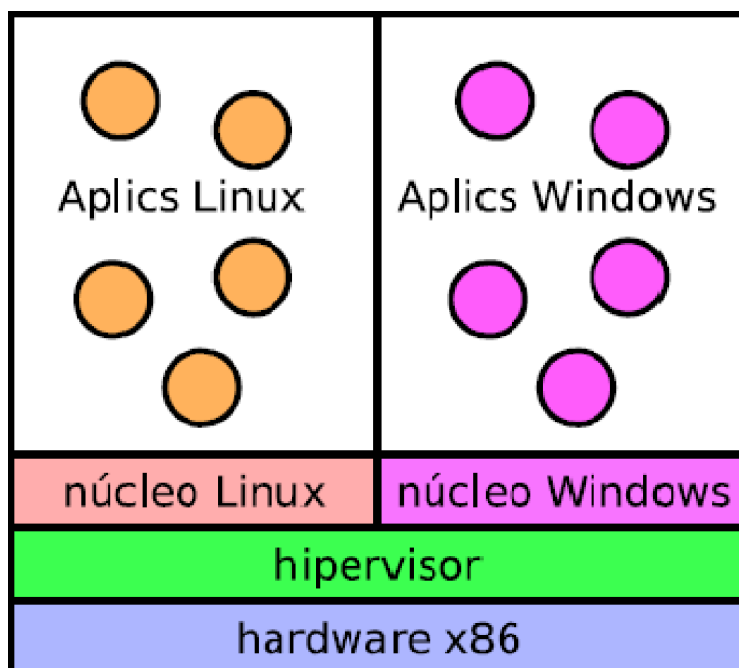


Fonte: Maziero (2017, p. 23).

Na Figura 4 foi apresentada uma aplicação java sendo executada sobre a máquina virtual do java (JVM), assim sendo, possível a execução desta aplicação java em diferentes plataformas de sistemas operacionais.

Máquinas virtuais de sistemas vem do inglês *System Virtual Machine* que se resume em um ambiente de inúmeras máquinas virtuais executadas sobre um único *hardware*, sendo assim, as máquinas virtuais são independentes possibilitando a execução de sistemas operacionais distintos. Este modelo foi elaborado para suportar virtualização de sistemas operacionais por completo, com várias aplicações sendo executadas sobre estes sistemas operacionais virtualizados. Podem ser apresentados como exemplos os ambientes *VirtualBox*, *VMware* e o *KVM*. A Figura 5 ajuda a ilustrar um exemplo comum de máquina virtual de sistemas (BARROS, 2016).

Figura 5 - Máquina virtual de sistemas.



Fonte: Maziero (2017, p. 23).

Na Figura 5 foram apresentados os sistemas operacionais Linux e Windows sendo executados em cima de uma camada conceituada de hipervisor, possibilitando assim a execução dos dois sistemas operacionais por completo ao mesmo tempo.

2.4 Tipos de hypervisor

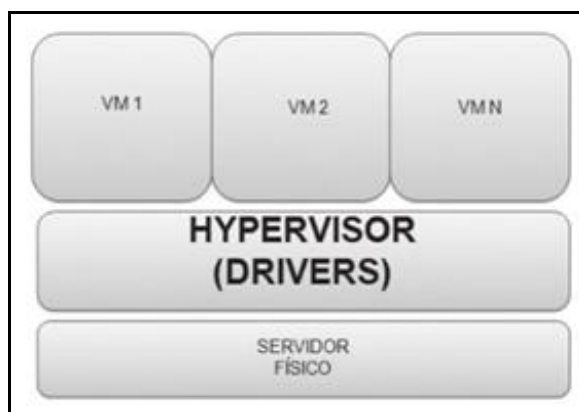
O *hypervisor* é a plataforma básica das máquinas virtuais. Entre suas principais funções estão o escalonamento de tarefas no processador, gerenciamento da memória e a manutenção dos estados das máquinas virtuais. A qualidade de um *hypervisor* pode ser definida pelo seu desempenho e pela sua escalabilidade (VERAS, 2016). Também é desejável que um hypervisor possua segurança sobre os recursos virtualizados e também agilidade na reconfiguração de recursos computacionais, sem haver interrupção nas operações do servidor de máquinas virtuais. Pode ser dividido em dois conceitos de plataforma sobre a qual o *hypervisor* é executado. São eles: tipo 1 e tipo 2. (GOLDBERG, 1973; KING, 2003).

2.4.1 Tipo 1

O modelo de virtualização tipo 1 ou *Bare-Metal* foi um termo dado à execução do *hypervisor* diretamente sobre um *hardware* físico, sem nenhuma outra camada de *software* abaixo dele, ou seja, neste tipo o hypervisor tem controle total sobre o processador e o restante do *hardware*. Dentro do tipo de virtualização *bare metal* existem dois modelos de *hypervisors* de acordo com Veras (2011): o *hypervisor* monolítico e o *hypervisor microkernel*.

O *hypervisor* monolítico (FIGURA 6) precisa de uma grande quantidade de código entre os recursos de *hardware* e as máquinas virtuais, pois este tipo de *hypervisor* emula todo o *hardware* físico para as máquinas virtuais. Neste modelo os *drivers* estão no próprio *hypervisor* (VERAS, 2016).

Figura 6 - Modelo de virtualização tipo 1 com hypervisor monolítico.

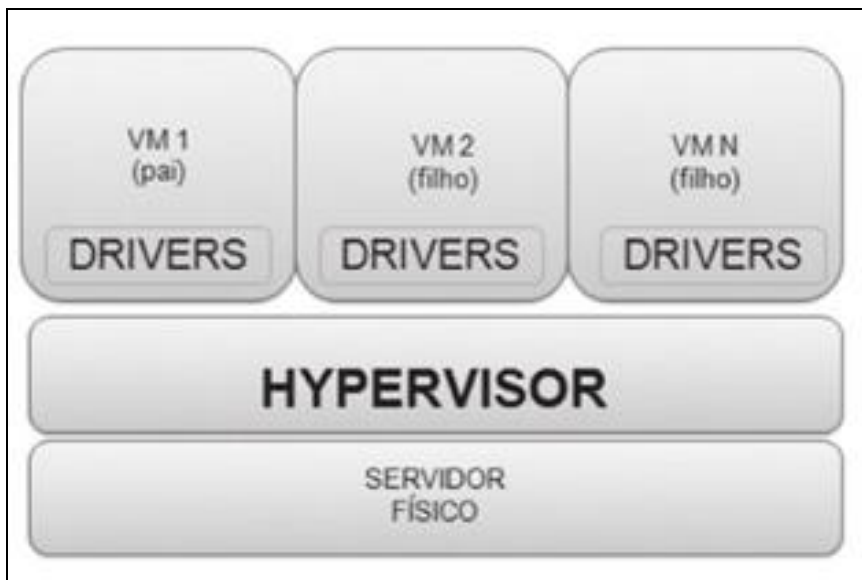


Fonte: Veras (2016, p. 102).

Na Figura 6 foi apresentado o modelo de virtualização com um *hypervisor* monolítico. Sendo assim, os *drivers* responsáveis pelas chamadas de sistemas estão alocados dentro do próprio *hypervisor* (VERAS, 2016).

O *hypervisor microkernel* (FIGURA 6) utiliza os *drivers* na própria máquina virtual e a única camada entre o sistema operacional convidado e o *hardware* físico é o próprio *hypervisor*. Os *drivers* estão instalados na máquina virtual. Utilizando este modelo de *hypervisor* se obtém um aumento na segurança devido a sua superfície de ataque ser menor (VERAS, 2016).

Figura 7 - Modelo de virtualização tipo 1 com *hypervisor* microkernel.



Fonte: Veras (2016, p. 102).

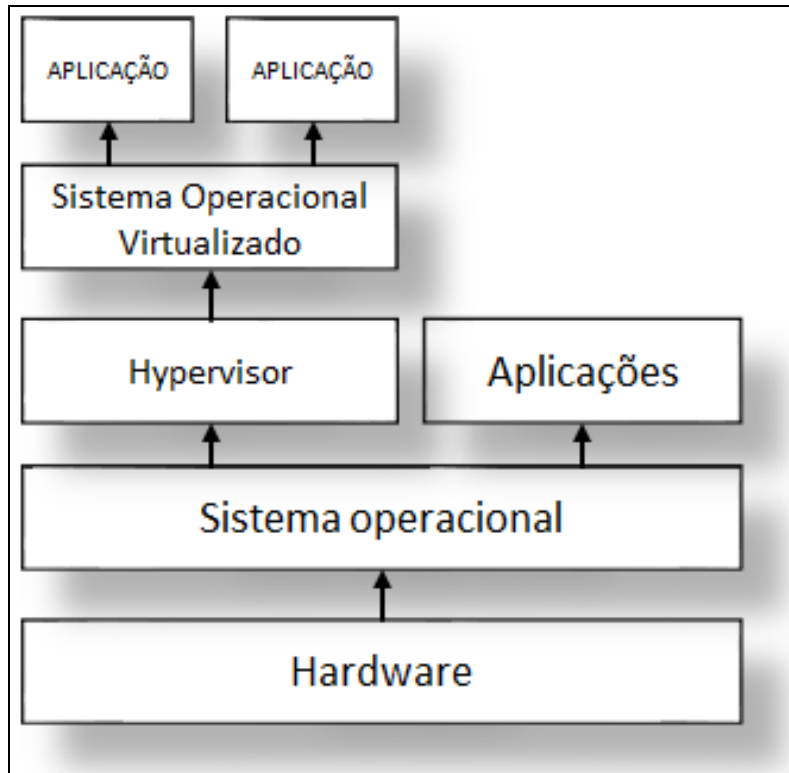
Na Figura 7 foi apresentado o modelo de virtualização com um *hypervisor microkernel*. Sendo assim, os *drivers* responsáveis pelas chamadas de sistemas estão alocados dentro da própria máquina virtual diferente de um *hypervisor* monolítico.

2.4.2 Tipo 2

O *Tipo 2* ou *hosted* que vem do português hospedeiro é um termo dado quando o *hypervisor* que é executado como um processo sobre um sistema operacional não virtualizado (FIGURA 8). Diferente da virtualização do tipo 1 este modelo não tem acesso direto aos recursos do *hardware* físico, pois existe uma camada de sistema operacional entre o *hardware* e o *hypervisor*. Podem ser

apresentados como exemplos o *Microsoft Virtual Server*, *VMware Workstation*, *VMware Server*, *Oracle VirtualBox*, *oVirt*, *VMware Player*, *Microsoft VirtualPC* e o *QEMU* (BARROS, 2016).

Figura 8 - Modelo de virtualização tipo 2.



Fonte: Do autor (2017).

Na Figura 8 foi apresentado um exemplo do modelo de virtualização tipo 2. Como demonstrado existe uma camada de sistema operacional entre o *hypervisor* e o *hardware*, sendo assim dificulta o acesso ao processador e memória do *hardware* físico, pois todas solicitações do sistema operacional virtualizado devem ser analisadas pelo sistema operacional real antes de encaminhar para o *hardware* físico (VERAS, 2010).

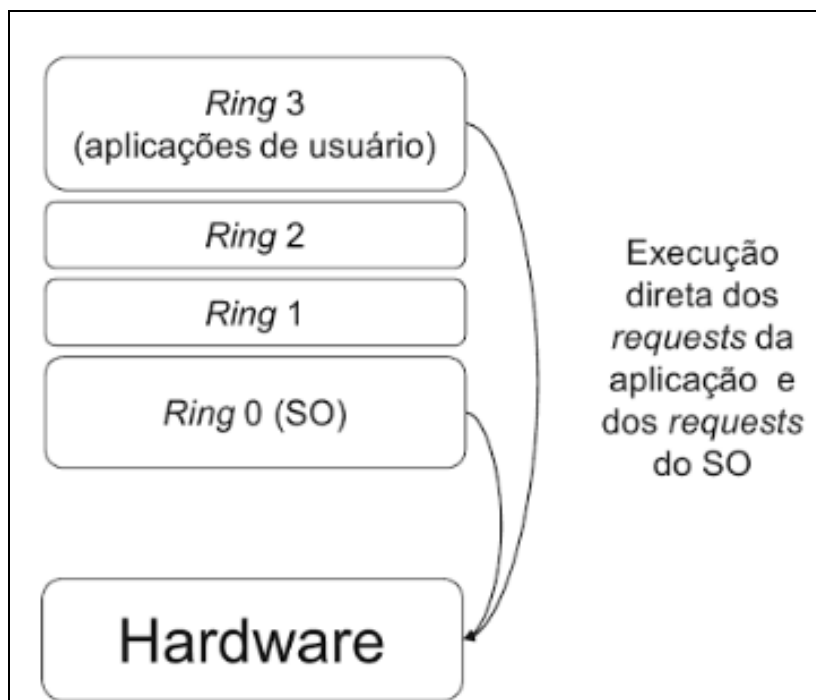
2.5 Modelos de virtualização

Neste subitem serão demonstrados os principais modelos de virtualização, iniciando pela virtualização completa ou total e após pela paravirtualização.

2.5.1 Virtualização completa ou total

Ao contrário do que parece a virtualização pode ser mais complicada, pois exige uma grande atenção de como é efetuado o dimensionamento de recursos físicos. O grande problema consiste quando a máquina virtual executa as instruções privilegiadas conhecidas também pelo nome *System ISA*¹. Para se ter melhor entendimento precisa-se ter um pouco de conhecimento da arquitetura x86. A arquitetura x86 tem quatro modos de operação para efetuar controle do uso dos recursos do processador. Estes modos são listados de 0 a 3, denominados de anéis de proteção que vem do inglês *Rings* ou *Current Privilege Level* que pode ser reconhecido pela sigla CPL (VERAS, 2011). Na maioria dos ambientes são utilizados apenas dois modos que são o anel 0 que detém os maiores privilégios de execução, que normalmente é utilizado pelo próprio sistema operacional real e o anel 3 que detém o menor privilégio para execução de processos comuns de usuário como os próprios aplicativos do usuário. Caso o usuário tente efetuar uma instrução privilegiada diretamente no anel 0 ocorrerá uma exceção conceituada de *trap*, que deverá ser analisado e tratada adequadamente conforme a Figura 9 (VERAS, 2010).

Figura 9 - Hierarquia na arquitetura do processador x86.



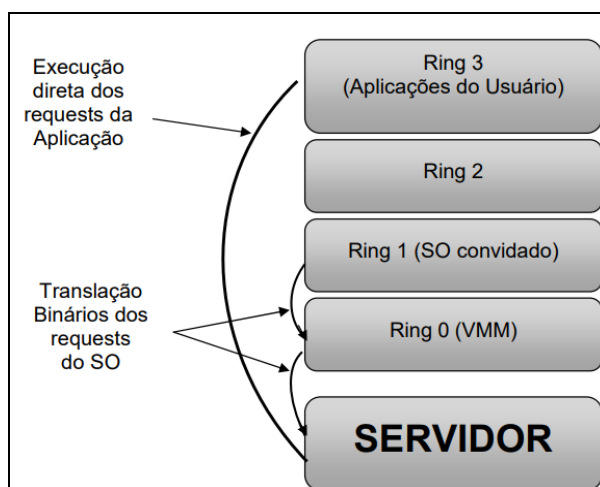
Fonte: Veras (2016, p. 55).

¹ *System ISA* seria a abstração dos processadores x86 através de um conjunto de instruções de máquina que faz uso da linguagem assembly.

Na Figura 9 foi apresentado como é o funcionamento da arquitetura do processador x86 sem a virtualização. Como demonstrado, as instruções privilegiadas e não privilegiadas podem ser executadas diretamente no *hardware*.

O Grande desafio em utilizar técnicas de virtualização nas arquiteturas x86 era como poderia ser efetuado o tratamento das instruções privilegiadas nos sistemas operacionais convidados sob um *hypervisor*. A empresa VMware conseguiu resolver este problema em 1998 desenvolvendo uma técnica de transação binária para possibilitar que o *hypervisor* seja executado a partir do anel 0, assim então, possibilitando a criação de máquinas virtuais. Desta forma houve também um aumento de desempenho nos sistemas operacionais convidados, pois o acesso ao *hardware* era mais rápido. Enquanto isso o sistema operacional foi movido para o anel 1 e por este motivo perde alguns privilégios referentes às instruções privilegiadas (VERAS, 2010). Desta forma foi possível utilizar a virtualização nas arquiteturas x86 conforme a Figura 10.

Figura 10 - Virtualização total na arquitetura x86.



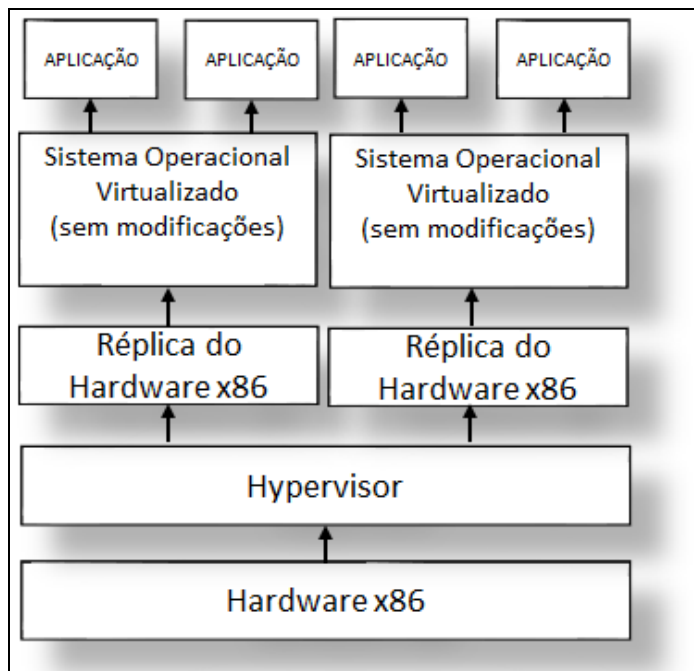
Fonte: Do autor, adaptado de Mattos (2008).

Na Figura 10 foi apresentada a arquitetura x86 dentro do modelo de virtualização total. Como demonstrado o *hypervisor* está sendo executado no anel 0, assim então possibilitando a execução de máquinas virtuais em cima de arquiteturas x86. Os *requests* citados na figura acima são as chamadas de sistemas (VERAS, 2010).

A virtualização total tem como objetivo disponibilizar uma réplica do *hardware* físico para a máquina virtual (FIGURA 11). Utilizando a virtualização total o sistema operacional convidado é executado sem haver modificações em suas chamadas de sistemas, ele trabalha desconhecendo totalmente que é um sistema virtualizado. Neste método de virtualização existem três inconvenientes (VERAS, 2016):

- I. Todas as instruções que são executadas pelo sistema operacional convidado serão testadas e tratadas adequadamente pelo *hypervisor*. Quando forem instruções não críticas, são executadas direto no *hardware* físico sem necessitar de um tratamento específico do *hypervisor* que está localizado neste modelo de virtualização no anel 0 e as instruções críticas vão ser executadas diretamente no *hypervisor*;
- II. A gerência de memória tem problemas técnicos relativos à implementação, pois cada máquina virtual foi implementada para ser executada em uma única instância. Por este motivo ocorre o inconveniente no uso da paginação de memória virtual, pois há uma grande disputa de recursos entre as instâncias de sistemas operacionais virtualizados, o que acarreta na queda do desempenho;
- III. Com a variedade de dispositivos de entrada e saída (E/S) a serem suportados pelo *hypervisor* o sistema operacional virtualizado necessita da criação de dispositivos genéricos, porém quando as máquinas virtuais utilizam os dispositivos genéricos de E/S ocorre uma perda real de desempenho no sistema operacional virtualizado.

Figura 11 - Virtualização total ou completa.



Fonte: Do autor (2017).

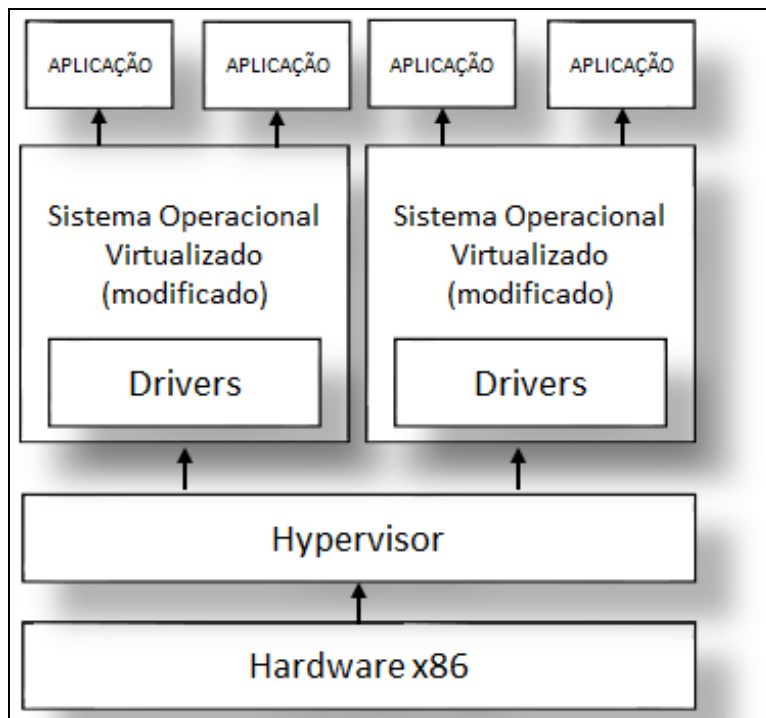
Na Figura 11 foi apresentado um exemplo do modelo de virtualização total. Como demonstrado o *hypervisor* efetua a criação de uma réplica total do *hardware* físico para as máquinas virtuais, assim possibilitando a virtualização sem haver modificações no sistema operacional virtualizado (VERAS, 2010).

2.5.2 Paravirtualização

A paravirtualização pode ser considerada uma medida de contorno para a virtualização total. A grande diferença é que os acessos aos recursos de *hardware* são utilizados através de *drivers* previamente instalados no próprio sistema operacional convidado, e por este motivo as chamadas de sistemas são alteradas. Desta forma os acessos aos recursos de *hardware* reais acontecem diretamente sem haver dispositivos genéricos virtualizando o *hardware* como no caso descrito na virtualização total (VERAS 2016).

De acordo com Veras (2016), o grande problema na paravirtualização seria que o sistema operacional convidado deverá ter conhecimento que é um sistema virtualizado, pois suas chamadas de sistemas padrão serão alteradas para ser efetuado uma separação automaticamente sem a necessidade de ser executados testes pelo *hypervisor* (FIGURA 12).

Figura 12 - Paravirtualização.

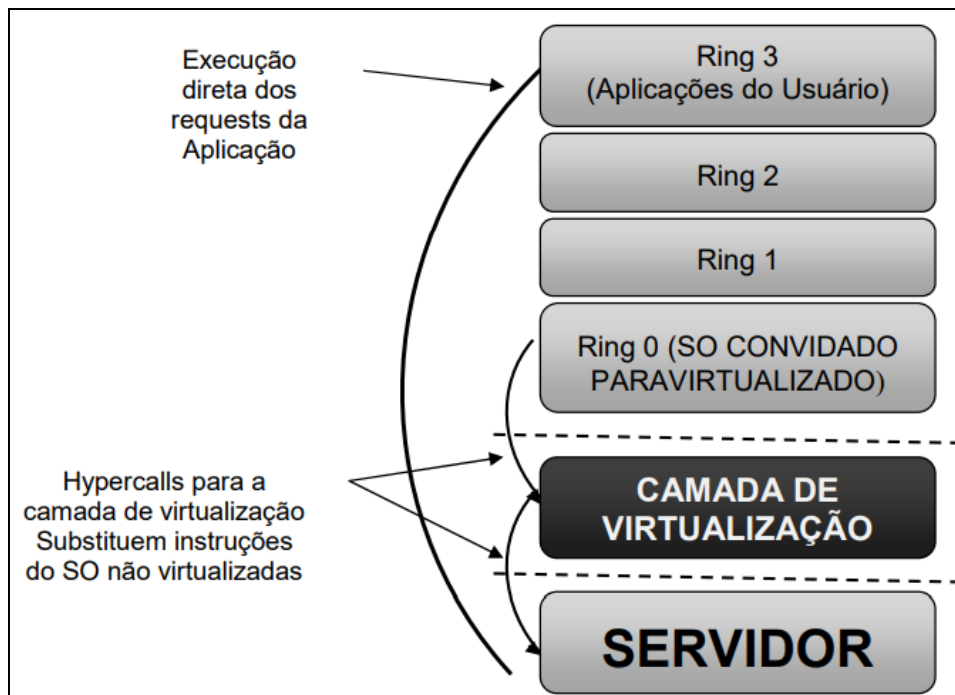


Fonte: Do autor (2017).

Na Figura 12 foi apresentado um exemplo do modelo de paravirtualização. Como demonstrado a paravirtualização necessita da instalação de *drivers* específicos dentro dos sistemas operacionais virtualizados, pois após a instalação desses *drivers* ocorre uma alteração no processo de chamadas de sistemas do sistema operacional (VERAS, 2010).

Segundo Veras (2016), o modelo de paravirtualização tenta corrigir os problemas da virtualização total permitindo que os sistemas operacionais virtualizados tenham o acesso direto ao *hardware* físico. Para que isso seja possível houve uma mudança na hierarquia do processador x86, pois é adicionada uma camada entre o anel 0 e o *hardware*, assim ganhando um melhor desempenho e eficiência nas instruções privilegiadas dos sistemas operacionais convidados (FIGURA 13). Para que isso ocorra com a paravirtualização, necessita da assistência de um compilador inteligente conceituado de hiper chamadas que vem do inglês *hypercalls*. Ele atua especificamente a substituição das instruções privilegiadas quando solicitadas pelos sistemas operacionais convidados. Este procedimento poupa um grande desempenho, quando comparado a virtualização total que utiliza a técnica de translação binária.

Figura 13 - Hierarquia do processador x86 na paravirtualização.



Fonte: Do autor, adaptado de Mattos (2008).

Na Figura 13 foi apresentada a arquitetura x86 dentro do modelo de paravirtualização. Como demonstrado o *hypervisor* está sendo executado em uma camada especialmente criada para ele, possibilitando assim usufruir dos recursos de *hardware* com um maior desempenho. Na paravirtualização as *hypercalls* substituem as instruções dos sistemas operacionais virtualizados. Os *requests* citados na figura acima são as chamadas de sistemas (VERAS, 2010).

2.5.3 Virtualização assistida por hardware.

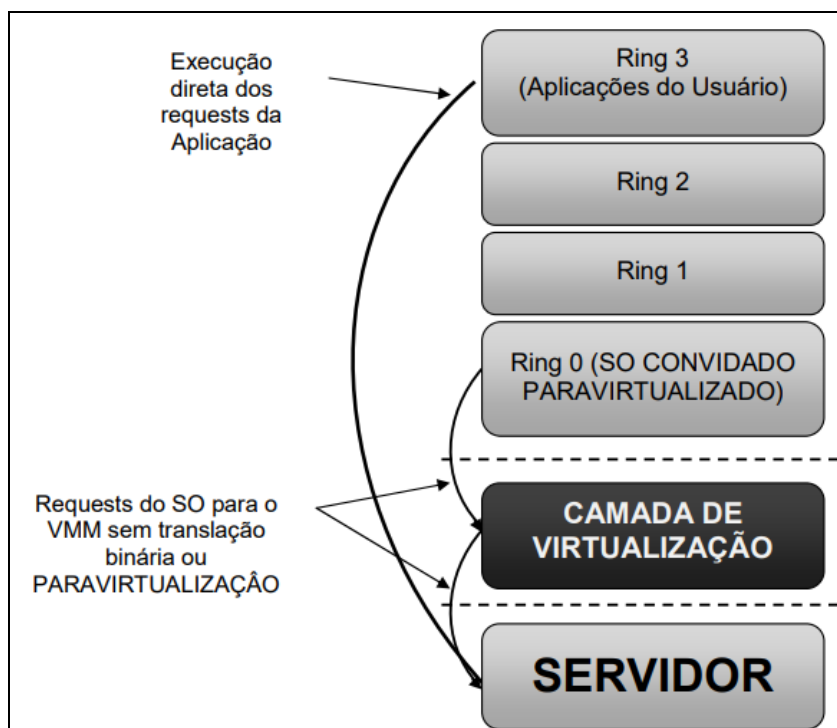
Este modelo de virtualização utiliza recursos que foram incorporados nas últimas versões de processadores da INTEL e da AMD. Estes recursos são conhecidos como Intel VT e AMD-V. As duas empresas fizeram um esforço para alterar o modo de como o processador se comporta com a virtualização. Estes recursos disponibilizados oferecem o funcionamento de sistemas operacionais convidados sem a necessidade de serem modificados com a instalação de *drivers* que é o caso da paravirtualização, ou também sem perder desempenho com a virtualização de *hardware* no caso da virtualização total (VERAS, 2010).

A grande diferença é que foi disponibilizado recurso de acesso direto ao *hardware* sem a necessidade de montar uma réplica ou modificar o sistema

operacional virtualizado. Esta mudança gera uma grande diferença em desempenho pois todas as instruções agora são capturadas e emuladas diretamente no próprio *hardware* sem a necessidade do *hypervisor* monitorar e tratar estas instruções. Assim, este modelo de virtualização satisfaz todos os critérios dos autores Popek e Goldberg, porém ainda existe uma única desvantagem que seria a necessidade do apoio explícito do processador (POPEK; GOLDBERG, 1974).

De acordo com Mattos (2008), ao ocorrer alguma instrução crítica na virtualização assistida por *hardware*, os processadores vão utilizar uma técnica conceituada pelo autor de “apanhar e emular” diretamente no próprio *hardware* físico sem a necessidade do *hypervisor* executar internamente as funções de monitoramento e tratamento de instruções. Na Figura 14 é ilustrada a hierarquia na arquitetura x86 da virtualização assistida por *hardware*.

Figura 14 - Virtualização Assistida por Hardware na Arquitetura x86.



Fonte: Do autor, adaptado de Mattos (2008).

Na Figura 14 foi apresentada a arquitetura x86 dentro do modelo de virtualização assistida por *hardware*. Como demonstrado, o *hypervisor* está sendo executado em uma camada especialmente criada para ele. Este conceito também é aplicado dentro da paravirtualização, porém neste modelo não serão utilizadas as chamadas *hypercalls* e também técnicas de tradução binária, pois existe o apoio

explícito do processador para as instruções e chamadas de sistemas. Os *requests* citados na figura acima são as chamadas de sistemas (VERAS, 2010).

2.5.4 Comparação entre os tipos de virtualização

De acordo com Veras (2016), a Tabela 1 faz uma comparação entre os modelos de virtualização. O autor efetuou a comparação entre as principais empresas no ramo de virtualização.

Tabela 1 - Comparação entre os modelos de virtualização.

	VIRTUALIZAÇÃO	TOTAL	VIRTUALIZAÇÃO ASSISTIDA POR HARDWARE	PARA VIRTUALIZAÇÃO
Técnica	Translação Binária e Execução Direta		Saída para modo raiz nas instruções privilegiadas	Hypercalls
Modificação do SO Virtualizado/Compatibilidade	SO virtualizado não modificado/exelente		SO virtualizado não modificado/exelente	SO virtualizado modificado/baixa compatibilidade
Desempenho	Bom		Considerável	Melhor em certos casos
Usado por	VMware, Microsoft, Parallels		VMware, Microsoft, Parallels, Xen	Vmware, Xen
Independência entre o SO virtualizado e o hypervisor	Sim		Sim	Não

Fonte: Do autor, adaptado de Veras (2011).

Na Tabela 1 foi apresentada uma comparação entre os modelos de virtualização. Conforme Veras (2011), não existe um modelo perfeito de virtualização, pois o resultado vai depender muito do ambiente aonde vai ser aplicado o conceito de virtualização.

2.6 Outros conceitos e soluções de virtualização

Estendendo o escopo da virtualização, serão demonstrados nesta seção será apresentado outros modelos de virtualização que são tão utilizados quanto a de virtualização de servidores, porém estes conceitos não têm a mesma visibilidade. Será apresentada a virtualização de redes, virtualização de *desktop* ou virtualização

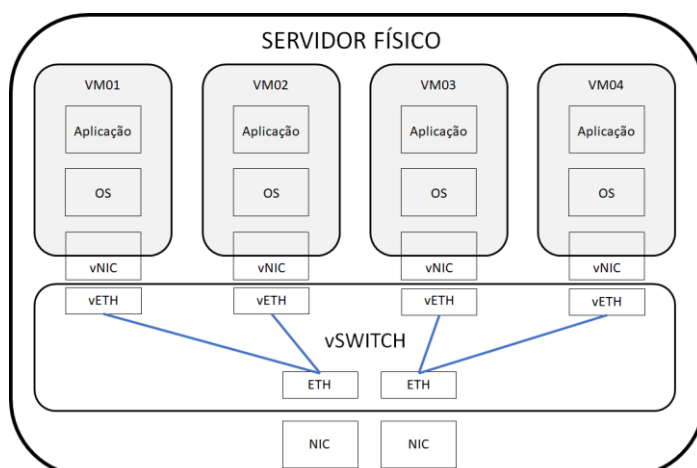
de estação de trabalho, virtualização de aplicativos, virtualização de *storage* ou virtualização de solução de armazenamento e containerização

2.6.1 Virtualização de rede

Os fabricantes de *softwares* de virtualização vêm trabalhando constantemente para diminuir a complexidade da utilização dos recursos de virtualização de rede. A empresa VMware resolveu estas dificuldades inicialmente implementando um *switch* por *software* conhecido como VSwitch que trabalha em conjunto com o *hypervisor*. Cada placa de rede virtual é chamada de vNIC e a função dela é conectar logicamente a máquina virtual ao VSwitch, assim possibilitando a troca de tráfego pela rede, porém esta configuração simples pode trazer vários inconvenientes que podem dificultar o gerenciamento da rede pois cada VSwitch passa a ser mais um ponto de configuração (VERAS, 2010).

A Cisco e a VMware criaram o conceito de um switch virtual distribuído (DVS) que possibilita o gerenciamento de todos os VSwitches a partir de um único ponto. A Cisco chamou esta nova funcionalidade de VN-Link. Esta melhoria permite a criação de vários links lógicos entre uma vNIC em uma máquina virtual e um *switch* com uma vNIC. A grande diferença é a criação de uma placa de rede virtual (vEth). Esta interface tem o mesmo conceito de uma placa física. Um VSwitch é criado para a VN-Link que pode implementar várias interfaces virtuais (vEth) por porta física e criar um mapeamento entre cada uma das portas virtuais que vai corresponder a uma Vnic na máquina virtual conforme a Figura 15 (VERAS, 2010).

Figura 15 - Comunicação entre a rede física e máquinas virtuais.

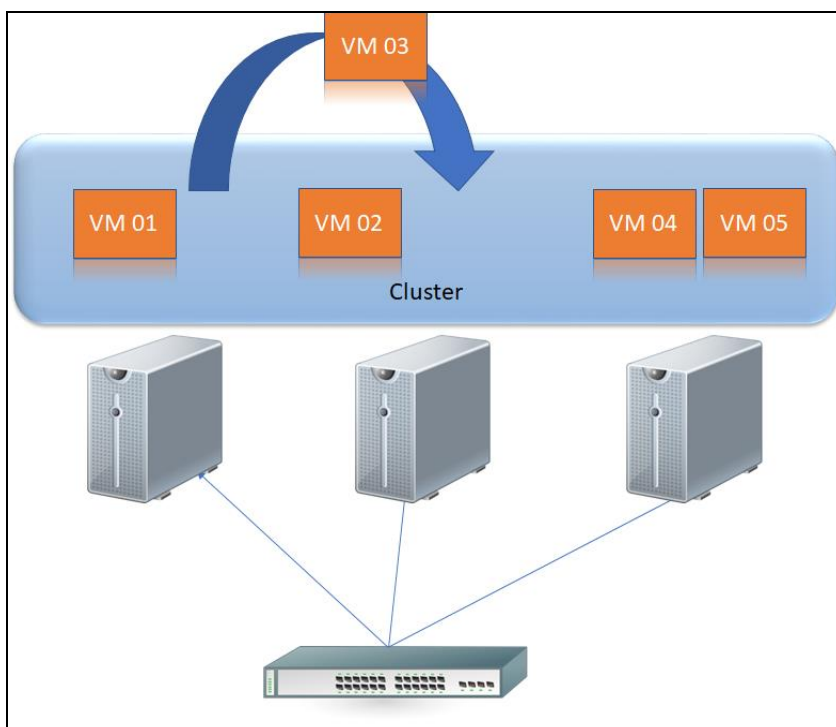


Fonte: Do autor (2017).

Na Figura 15 foi apresentada a arquitetura de um switch distribuído virtualmente. Como demonstrado, as placas de rede virtuais (vNIC) são conectadas logicamente em uma *network interface card* (NIC). Isto ocorre, pois, o switch virtual distribuído (DVS) interliga todo tráfego entre as placas de rede virtuais das máquinas virtualizadas e as placas de rede reais do servidor físico. Outro recurso do DVS é que o tráfego entre máquinas virtualizadas não passa pelas NIC do servidor físico, pois o tráfego de rede entre máquinas de virtualização é redirecionado dentro do próprio DVS (VERAS, 2010).

A utilização do recurso de virtualização de rede possibilita o funcionamento de várias técnicas de alto desempenho e alta disponibilidade dentro do cenário de virtualização como: balanceamento de carga, redundância, aumento na largura de banda, flexibilidade, movimentação de máquinas virtuais entre *hardwares* diferentes e entre outros (FIGURA 16). Para estas técnicas citadas funcionarem é necessário aplicar um conceito de *cluster* entre os servidores de *hypervisor*. O conceito de *cluster* será detalhado dentro do capítulo 3 (VERAS, 2010).

Figura 16 - Servidores de hypervisor em um cluster.



Fonte: Do autor (2017).

Na Figura 6 foi apresentado um cluster que está interligando os servidores de *hypervisor* através de um *switch* que utiliza o padrão ethernet para comunicação.

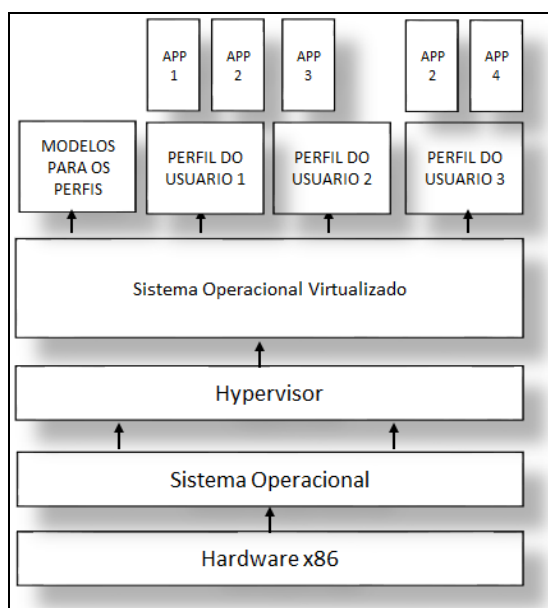
Neste cenário de cluster é possível migrar as máquinas virtuais entre os servidores de *hypervisor*.

2.6.2 Virtualização de desktop

A virtualização de desktop está sendo utilizada cada vez mais dentro das empresas. O objetivo da virtualização de desktop ou também conhecido pela abreviatura VDI que vem do inglês *Virtualization Desktop Infrastructure* não é o mesmo da virtualização servidor, ela permite ter uma maior mobilidade para os usuários e se ter uma grande rapidez quando se trata de mudanças no ambiente de TI (VERAS, 2016).

O conceito de virtualização de desktop é que o usuário consiga acessar sua estação de trabalho independentemente de onde ele estiver. Aplicando este conceito conseguimos ter uma mobilidade grande como já foi citado. A virtualização de desktop já é muito utilizada nos dias de hoje dentro das empresas. Pode-se citar como exemplo a própria função de terminal *server* aonde o usuário consegue acessar uma máquina remotamente. A Figura 17 consegue demonstrar de uma forma simples a funcionalidade da virtualização de desktop. Nesta figura é demonstra uma virtualização de sessenta desktops em um único servidor físico (BARROS, 2016)

Figura 17 - Conceito de infraestrutura de virtualização de desktops.



Fonte: Do autor (2017).

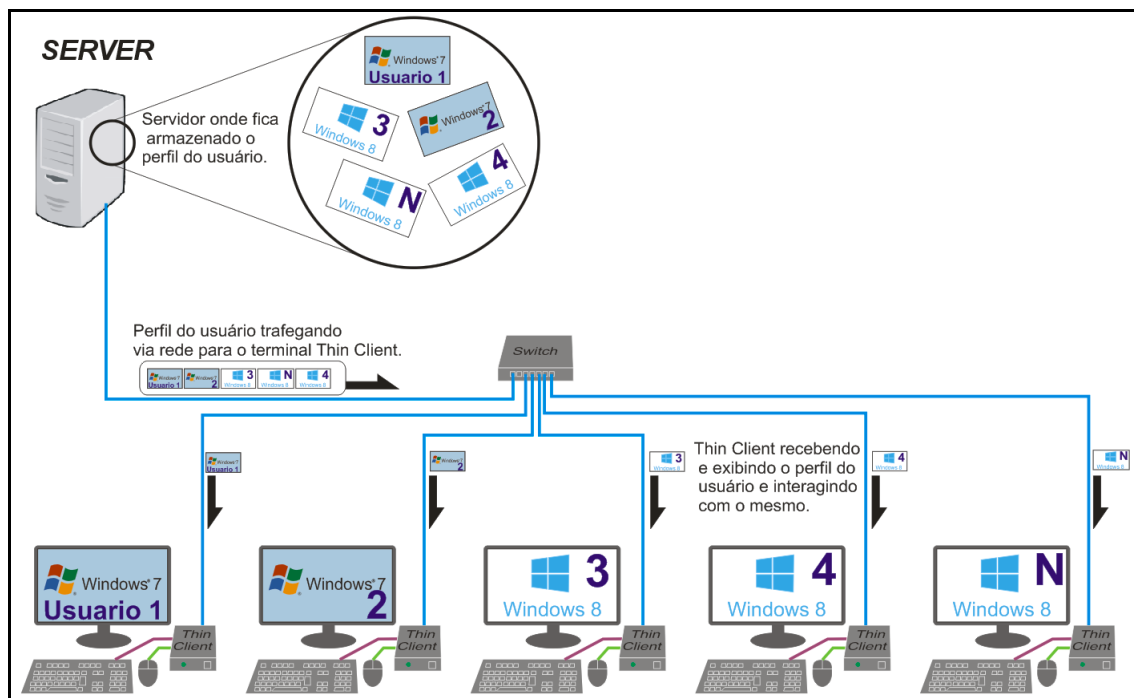
Na Figura 17 foi apresentado o conceito de virtualização de desktop utilizando uma infraestrutura típica de virtualização de servidores. A diferença no conceito de virtualização de desktop é a possibilidade de criar vários perfis de usuários conforme for necessário. Estes perfis são baseados em modelos e dentro de cada um deles contém configurações e aplicações específicas, assim facilitando a configuração de novas máquinas para diferentes tipos de usuários, pois todas as aplicações já estão previamente instaladas e configuradas dentro dos perfis (VERAS, 2010).

Os principais recursos das ferramentas de virtualização de estações de trabalho são:

- Gerenciamento simplificado e um grande controle sobre as estações dos usuários;
- A grande redução de custo nas manutenções e redução de custo na compra de estações novas;
- O aumento da produtividade para os usuários, pois a virtualização de desktops permite que os acessos sejam efetuados por quaisquer dispositivos como: *tablets*, smartphones, computadores e notebooks;
- Desktop está sempre ligado, o usuário não precisa necessariamente desligar, apenas desconectar e quando necessitar conectar novamente vai continuar com as atividades de onde ele parou;
- Aumento na segurança dos dados das empresas, já que os desktops ficam alocados dentro dos próprios servidores da empresa ou também em uma nuvem privada, o que facilita a execução de cópias de segurança dos arquivos dos usuários (*backup*) (BARROS 2016).

Segundo Ben-Shaul (2011), o conceito de virtualização de desktop pode ser aplicado em *Thin Client*, que normalmente utilizam o protocolo de comunicação *Remote Desktop Protocol* (RDP) proprietário da Microsoft, porém existem outros protocolos que podem ser utilizados como o *Independent Computing Architecture* (ICA), o *Virtual Network Computing* (VNC) entre outros (FIGURA 18).

Figura 18 - Utilização de Thin Client no conceito de virtualização de desktop.



Fonte: Do Autor, adaptado de Barros (2016).

Na Figura 18 foi apresentado um cenário que faz uso do conceito de virtualização de desktop em dispositivos *thin client*. Neste cenário o perfil e arquivos do usuário estão sendo armazenados totalmente dentro de um servidor. Com isso o dispositivo *thin client* apenas recebe a imagem do seu perfil em tempo real deixando assim o todo consumo de memória, disco e processamento para o servidor (VERAS, 2010).

2.6.3 Virtualização de aplicação

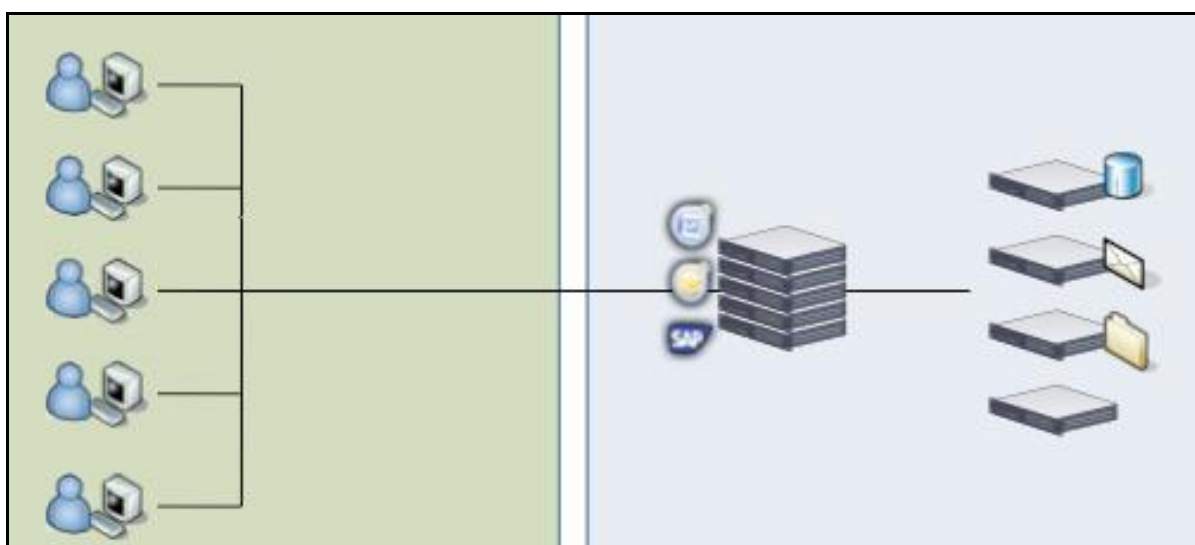
Segundo Rocha (2013), a primeira empresa que desenvolveu a técnica de virtualização de aplicação foi a empresa americana Citrix System, que apresentou um produto capaz de virtualizar as aplicações em diferentes sistemas operacionais.

De acordo com Rocha (2013), a virtualização de aplicações cria a possibilidade de acessar e utilizar as aplicações de uma forma totalmente remota, sem a necessidade de realizar a instalação dos aplicativos virtualizados nas máquinas locais. No início, a utilização desta tecnologia só era possível dentro de uma rede local "LAN", porém com a evolução da tecnologia de virtualização de aplicação, atualmente, já se tornou possível o acesso via internet "WAN", de forma

fácil, simples e segura, com possibilidades dos acessos serem efetuados através de smartphones ou tablets.

Para utilização do conceito de virtualização de aplicação deverão ser disponibilizados servidores onde serão centralizados todos os aplicativos virtualizados. A partir deste servidor a tecnologia de virtualização de aplicativos se encarrega de fazer a entrega para os usuários, seja em uma rede LAN ou WAN de forma segura e performática conforme a Figura 19 (ROCHA, 2013).

Figura 19 - Virtualização de aplicações.



Fonte: Do autor, adaptado de Rocha (2013).

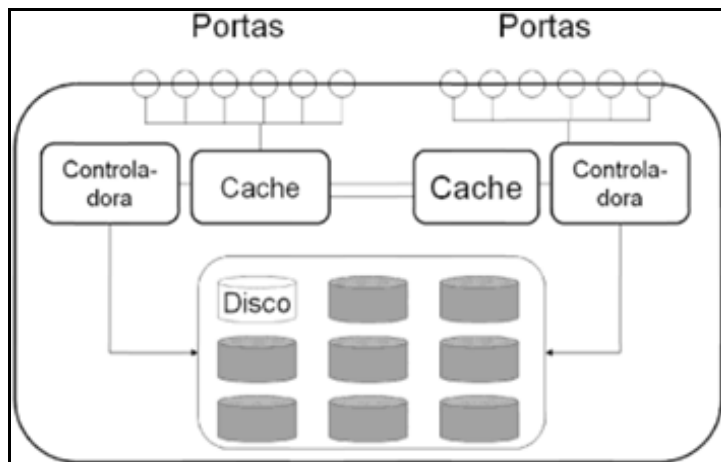
Na Figura 19 foi apresentado um cenário que faz uso do conceito de virtualização de aplicação. Conforme demonstrado na figura acima todo o processamento é realizado nos servidores onde foram instalados e configurados os aplicativos, desta forma o acesso é rápido e bastante seguro, tornando possível o acesso de aplicativos em diferentes sistemas operacionais. Pode ser utilizado como exemplo o acesso do pacote de aplicativos Microsoft Office em dispositivos como Android, IOS, Linux e ThinClient (ROCHA, 2013).

2.6.4 Virtualização de *storage*

De acordo com Veras o *storage* é um componente crítico na infraestrutura de TI, pois é responsável direto pelo armazenamento entregue aos servidores do *datacenter*. O *storage* tem a função de administrar e armazenar toda a entrada/saída (I/O) de dados dos servidores (VERAS, 2011).

Diferente de um servidor de arquivos, o *storage* pode ser visto como um servidor de vários discos de armazenamentos. Um servidor quando conectado a um *storage* apenas visualiza o disco e utiliza do sistema de arquivos do próprio sistema operacional conforme a Figura 20 (VERAS, 2016).

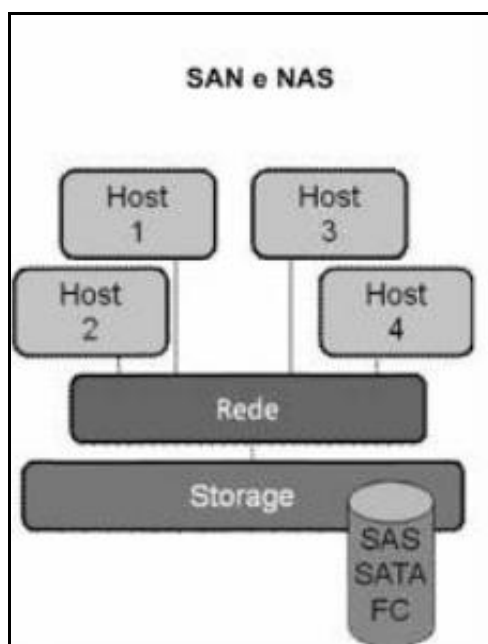
Figura 20 - Arquitetura de um *storage*.



Fonte: Veras (2011, p. 140).

Atualmente os *storages* estão utilizando os conceitos de *Storage Area Network (SAN)* ou o *Network Attached Storage (NAS)*. Estes conceitos permitem o acesso aos *storages* a partir da rede *ethernet* conforme Figura 21 (VERAS, 2016).

Figura 21 - Técnicas de entrada/saída.



Fonte: Veras (2011, p. 140).

Segundo Veras (2010), o armazenamento modelo SAN é baseado no padrão *Fiber Channel* ou *Gigabit Ethernet*. O acesso a dados é de baixo nível, isto porque o servidor solicita por blocos específicos ou segmentos de dados de discos específicos. O SAN apenas disponibiliza o armazenamento liberando o sistema de ficheiros a cargo do cliente.

Já o modelo NAS é baseado apenas no padrão *Gigabit Ethernet*. Ele é dedicado ao armazenamento de ficheiros dentro de uma rede. Basicamente o NAS tem o objetivo único de fornecer serviços de armazenamento de dados a outros dispositivos. A grande diferença é que o modelo NAS oferece um sistema de arquivo, sendo assim esta a principal diferença destes dois (VERAS, 2010).

2.6.5 Containerização

De acordo com Yu (2007), a virtualização a nível de sistema operacional é definida como a existência de múltiplas instâncias totalmente isoladas de espaços de usuário, que são gerenciadas pelo *kernel* deste sistema. Estas instâncias, chamadas de *container* representam todo um ambiente de execução: uma aplicação, juntamente com todas suas dependências, bibliotecas e arquivos de configuração, agrupados em um único pacote. Ao criar um *container* de uma determinada aplicação e suas dependências, a diversidade de sistemas operacionais e infraestruturas subjacentes são abstraídas.

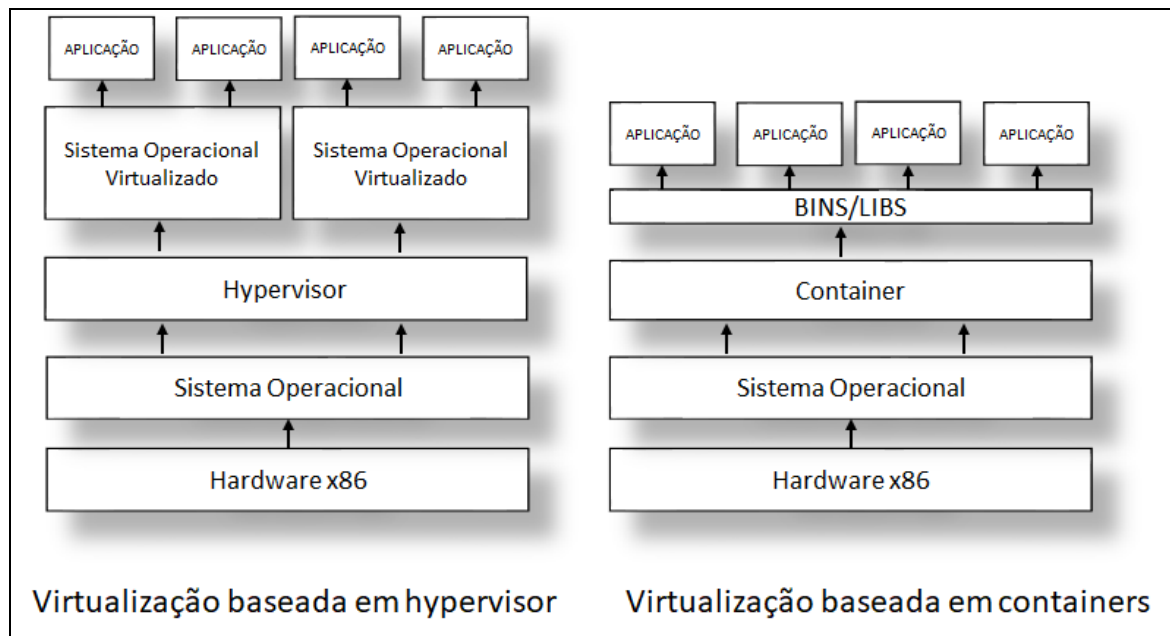
Este modelo permite a criação de partições lógicas de uma forma que cada uma destas partições seja totalmente isolada, isto tudo compartilhando o mesmo *kernel* de um único sistema operacional. Este modelo de virtualização permite em um único sistema operacional nativo executar múltiplas instâncias totalmente isoladas. Ele é muito utilizado nos sistemas operacionais Linux para virtualização de aplicações propriamente ditas (CORRÊA 2016).

A vantagem de utilizar *containers* é que não é necessário existir um sistema operacional para virtualizar uma aplicação. Sendo assim é obtida uma grande otimização de recursos computacionais se comparado aos modelos tradicionais de virtualização tipo *hosted* e tipo *bare-metal*. Entretanto, em cada *container* continua existindo uma camada composta por arquivos binários conhecidos como Bin e também por bibliotecas específicas chamadas de Lib (FIGURA 20). O diretório Bin

contém dentro dele alguns arquivos binários que são executáveis e que armazenam alguns comandos padrões do sistema linux como o exemplo apt-get install (BARROS, 2016).

Máquinas virtuais típicas e os *containers* possuem funcionalidades muito similares levando em consideração o seu isolamento e sua alocação de recursos computacionais. No entanto, a abordagem da arquitetura é diferente e a comparação entre as duas arquiteturas é apresentada na Figura 22. Esta figura também ajuda a entender a diferença entre a máquina virtual e um *container* (BARROS, 2016).

Figura 22 - Comparação da estrutura máquinas virtuais típicas e containers.



Fonte: Do autor (2017).

Na Figura 22 foi apresentada uma comparação entre a arquitetura de *containers* e a arquitetura de máquinas virtuais. Nesta figura também é demonstrado um *container* com quatro aplicações sendo executadas sobre ele. Desta forma as aplicações são executadas em um único sistema operacional, porém compartilhando o *kernel* do sistema operacional nativo entre si. Isso faz com que os *containers* sejam mais leves quando comparados a uma máquina virtual. A parte compartilhada do *kernel* é somente leitura, enquanto que cada *container* possui o seu espaço específico para escrita (BARROS, 2016).

No entanto, conforme Rubens (2015), *containers* e máquinas virtuais podem ser vistas entre si como tecnologias complementares ao invés de competitivas. Isso

se deve ao fato de ser possível rodar *containers* em máquinas virtuais, facilitando, através da virtualização de *hardware*, a gerência da infraestrutura (tais como rede e armazenamento), além de aumentar o isolamento e a segurança.

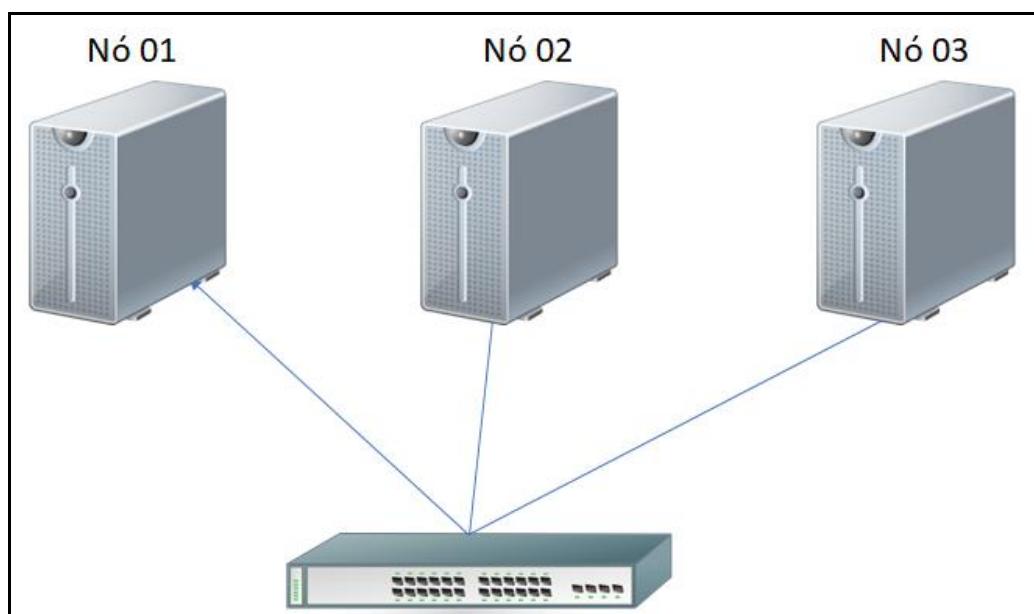
2.7 Cluster

Segundo Alecrim (2013), o conceito de *cluster* significa “aglomerar” ou “aglomeração” e pode ser aplicado em vários contextos. No caso da computação, o termo define que uma arquitetura de sistema é capaz de combinar inúmeros computadores para trabalharem em equipe.

Cada computador que compõem um cluster recebe o nome de nó que vem do inglês *node*. Não existe um número máximo de nós para um cluster, mas independentemente do número de nós que compõem o cluster, ele deve ser totalmente transparente para os usuários finais ou para um sistema que necessita do processamento de um cluster como um único computador (ALECRIM, 2013).

De acordo com Alecrim (2013), nós do *cluster* necessitam estar interconectados uns aos outros. Atualmente o protocolo mais comum para comunicação entre os nós do *cluster* é o padrão *ethernet* (FIGURA 23).

Figura 23 - Cluster interligado pela rede ethernet.



Fonte: Do autor (2017).

Na Figura 23 foram apresentados três servidores em *cluster* que estão conectados em um *switch*. Este equipamento faz o uso do protocolo de rede *ethernet* para receber e enviar todo tráfego de rede (ALECRIM, 2013).

De acordo com Veras (2009), a utilização do *cluster* se mostra uma solução viável de se implementar, pois os nós que compõem o *cluster* não precisam ser necessariamente um *hardware* específico para *clusters*, pode ser utilizado vários computadores comuns. Outra vantagem na solução de *cluster* que é citado pelo autor, seria a possibilidade de utilizar *hardwares* totalmente homogêneos dentro de um cluster, porém é importante que todos os nós que compõem o *cluster* utilizem o mesmo sistema operacional, isto para que *software* que gerencia o *cluster* consiga garantir que todos os nós se integrem uns aos outros.

Atualmente, existem vários modelos de *cluster*, mas os principais modelos são: *cluster* de alta disponibilidade, *cluster* de alto desempenho e *cluster* de balanceamento de carga.

- *Cluster* de alta disponibilidade tem como seu foco principal deixar a aplicação sempre em pleno funcionamento. Neste modelo não é aceitável que a aplicação ou sistema pare de funcionar por um grande espaço de tempo;
- *Cluster* de alto desempenho são direcionados para aplicações que necessitam de um grande processamento;
- *Cluster* para balanceamento de carga é utilizado quando existe a necessidade de distribuir as tarefas igualmente entre os nós de um *cluster* (SILVA, 2012).

Na próxima seção será apresentada a metodologia de trabalho do estudo.

3 METODOLOGIA

Segundo Junior (2011), a metodologia científica envolve argumentos teóricos e fundamentados a um referencial em relação à ciência, que detalha as pesquisas, relatórios gerados e posteriormente a coleta de dados, análise e interpretação de dados gerados, organização de dados, elaboração de relatórios e concepção do objeto. Neste aspecto, o atual capítulo do trabalho tem como objetivo exibir a metodologia utilizada para alcançar os objetivos propostos.

Todas as regras metodológicas utilizadas para direcionar o trabalho segue o Manual da Univates para Trabalhos Acadêmicos (CHEMIN, 2015).

Os próximos passos deste capítulo referem-se a demonstrar a metodologia utilizada, coleta de dados, análise dos resultados, sugestões de melhorias e sugestões de implementações. Os tópicos a seguir detalham estas questões citadas acima.

3.1 Método de pesquisa e trabalho

Segundo Marconi e Lakatos (2008), o conhecimento científico se diferencia dos demais por se tratar de fatos concretos, pois suas suposições são testadas e a veracidade colocada à prova através de experiências. Por este motivo é imprescindível que todo o levantamento teórico obtido até o momento sirva de suporte para o estudo do cenário real, além de ser necessário para que implementações práticas sejam feitas de maneira correta e adequada.

3.1.1 Quanto aos objetivos

Em relação a propósitos coletivos, considera-se como exploratório no sentido que há o interesse de se familiarizar com o ambiente do problema (GIL, 2010).

Com base no objetivo da pesquisa, procurar-se-á identificar entre duas soluções de virtualização a melhor quando se trata de consumo de recursos computacionais. Sendo que será avaliado o processamento, consumo de memória, IO e tráfego de rede.

3.2 Procedimentos de pesquisa

Nesta seção serão explicados os principais procedimentos de pesquisa utilizados no projeto.

3.2.1 Referencial teórico

O referencial teórico é uma espécie de entendimento científico. É através dele que foi feito a base sobre os temas abordados, além de ser o ponto de início para o projeto (JUNIOR, 2011).

Os temas estudados foram separados de acordo com alguns tópicos referenciados nas principais bibliografias dos principais autores de títulos sobre o assunto.

3.2.2 Coleta de dados

Nesta etapa se aplicam as técnicas escolhidas para o levantamento dos dados previstos (MARCONI, LAKATOS, 2008).

No contexto do trabalho, os esforços para coleta de dados estarão focados na análise de consumo dos recursos computacionais de servidores que fazem uso das técnicas de virtualização típicas e *containers*.

3.2.3 Análise de dados

Haverá uma seção reservada à demonstração dos resultados apoiada nos métodos de coleta empregados (GIL, 2010).

Após conseguir os resultados das coletas, será efetuada uma análise e logo após a conclusão dos dados extraídos (MARCONI, LAKATOS, 2008).

Desta forma, os dados coletados serão analisados e demonstrados em tabelas, gráficos e quadros. As sugestões de melhorias serão propostas com base nos resultados retirados nesta fase do projeto.

4 TRABALHOS RELACIONADOS

Com a finalidade de verificar a viabilidade do proposto trabalho, foi realizada uma busca por trabalhos semelhantes. Foi localizado um único trabalho efetuando a comparação entre máquinas virtuais e *containers*.

Nesta seção será resumido este trabalho comparativo. O estudo desse artigo auxiliou na definição de uma metodologia para realizar os testes e obter uma prévia dos resultados desta comparação.

4.1 Comparação de desempenho de máquinas virtuais e containers

Neste artigo Felter et al. (2014), exploraram o desempenho de implantações de máquinas virtuais tradicionais e compararam com o conceito de containerização. Nesta comparação foi utilizado um conjunto de cargas de trabalho para aumentar a utilização da CPU, memória, armazenamento e recursos de rede.

No cenário de teste foi utilizado o *software* de virtualização KVM para representar um *hypervisor* típico e para representar o conceito de *containers* foi utilizado o *software* Docker.

O objetivo deste artigo segundo os autores foi de aplicar uma carga de trabalho em máquinas virtuais e logo após introduzir a mesma carga de trabalho em *containers* e desta forma obter uma comparação da utilização dos recursos computacionais entre os dois modelos propostos.

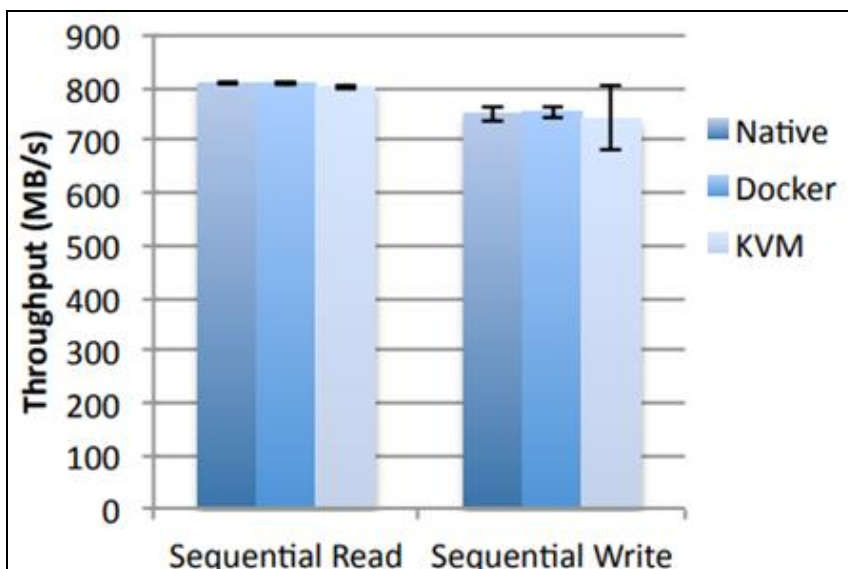
Todos os testes executados pelos autores foram realizados em um servidor IBM System x3650 M4 com dois processadores Intel Sandy Bridge-EP Xeon E5-2665 2,4-3,0 GHz para um total de 16 núcleos e 256GB de memória RAM. Foi utilizado para sistema operacional Ubuntu 13.10 de 64 bits com o *kernel* Linux 3.11.0. Já a versão utilizada do Docker foi a 1.0. Por razões de consistência os autores utilizaram a versão Ubuntu 13.10 em todos os *containers* do Docker e nas máquinas virtuais também foi utilizado a mesma versão.

Foi utilizado o *software microbenchmarks* para conseguir mensurar o consumo de CPU, memória, rede e armazenamento.

4.1.2 Testes executados

Para obtenção de dados os autores aplicaram diferentes testes para cada componente do servidor. No caso da utilização do armazenamento foi efetuado um teste de leitura e gravação sequencial utilizando em média 60 segundos de intervalo (FIGURA 24).

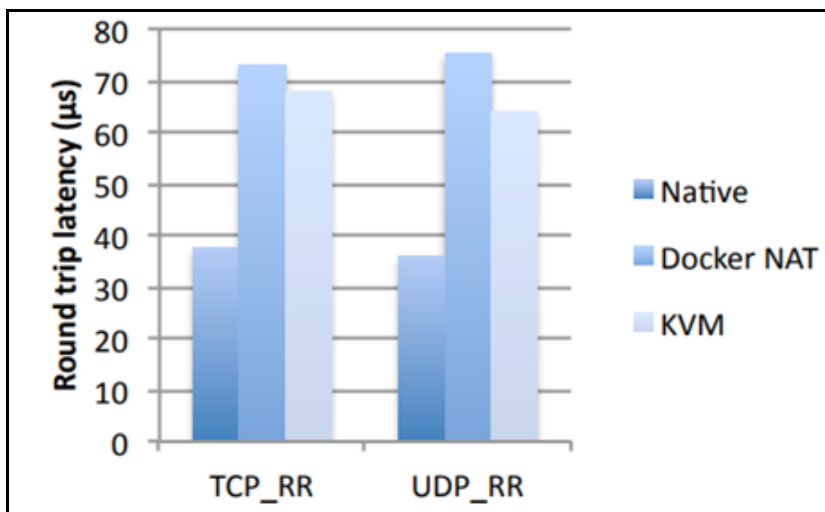
Figura 24 - Testes de leitura e gravação em disco.



Fonte: Felter, *et al.* (2014).

Já na obtenção dos dados para utilização de rede foi efetuada o teste onde um cliente enviava um pedido de 100 *bytes* para o servidor e logo após o servidor enviava uma resposta de 200 *bytes* (FIGURA 25).

Figura 25 - Testes de rede.



Fonte: Felter *et al.* (2014).

Segundo os autores, em geral, o Docker se iguala ou excede o desempenho do KVM em todos os casos que foram testados. Os resultados também apontam que tanto o KVM quanto o Docker, ao serem introduzidas pequenas cargas de trabalho eles consomem os recursos computacionais de formas muito similares. Já para cargas de trabalho intensas, ambos modelos de virtualização devem ser utilizados com cuidado. Os autores também ressaltaram que o desempenho do KVM melhorou consideravelmente desde a sua criação.

5 DESENVOLVIMENTO

Neste capítulo será apresentado o desenvolvimento do projeto e as tecnologias utilizadas no protótipo de coleta e análise dos dados.

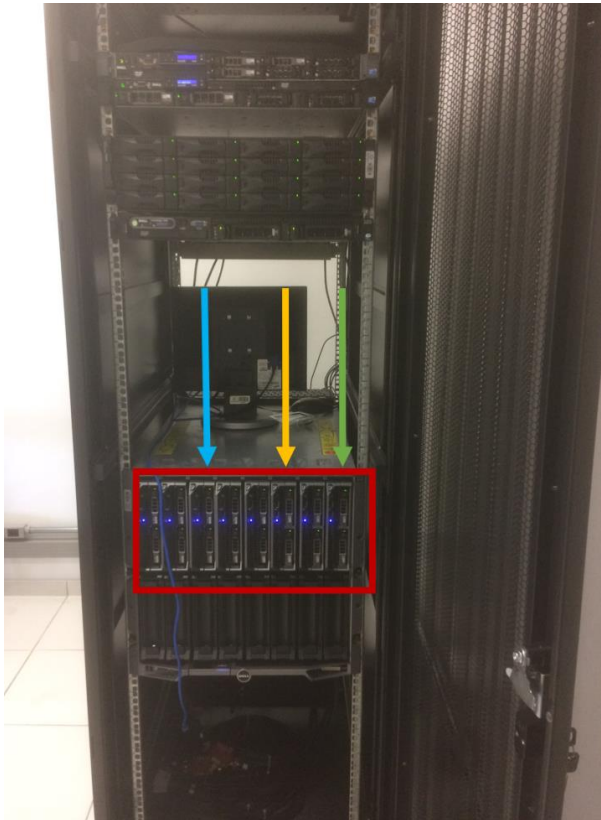
5.1 Implementação dos cenários de testes

Os Hardwares utilizados para efetuar os testes propostos foram disponibilizados pela própria instituição de ensino Univates. Foram disponibilizados três hardwares de servidores Dell, sendo do modelo PowerEdge M710HD. Estes hardwares continham um processador Intel Xeon CPU E5-2450, disco rígido de 120 SAS 12Gbps e memória de 130GB (FIGURA 25).

Estes equipamentos foram nomeados de SRV001, SRV002 e SRV003. O desenvolvimento do projeto foi dividido em duas partes, sendo eles o cenário 1 e o cenário 2. O cenário 1 foi implementado a tecnologia de containerização utilizando a ferramenta Docker. Já o cenário 2 foi implementado a tecnologia de virtualização tradicional utilizando máquina virtual do tipo 1 com uso software VMware ESXI na versão 6.

Os sistemas operacionais utilizados neste projeto tanto no cenário 1 quanto no cenário 2 será o Linux Ubuntu 18.04 para fins de padronização. Na figura 25 será ilustrado os servidores por com setas coloridas, sendo a seta da cor amarela o servidor SRV002, a seta da cor verde o servidor SRV001 e a seta azul o servidor SRV003.

Figura 25 - Estrutura física dos servidores utilizado nos testes.



Fonte: Do autor (2018).

Na figura 25 é demonstrado o cenário físico real cedido pela instituição de ensino Univates utilizada nos testes de containerização e virtualização propostos neste trabalho.

5.1.1 Cenário 1

Conforme descrito acima o cenário 1 (FIGURA 26) foi implementado a tecnologia de containerização com a ferramenta Docker para auxiliar na criação containers.

Figura 26 - Estrutura lógica do cenário 1.



Fonte: Do autor (2018).

A figura 26 ilustra a estrutura lógica do cenário 1. Conforme demonstrado na ilustração foi instalado um sistema operacional Ubuntu nativo no hardware físico. Sobre o sistema operacional é configurado o Docker. Para efetuar a instalação do software Docker é necessária uma série de comandos que serão listados na figura 27.

Figura 27 - Lista de comando para instalação do Docker.

```

1  sudo apt-get update
2
3  sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80
4  --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
5
6  sudo apt-add-repository 'deb https://apt.dockerproject.org/repo ubuntu-xenial main'
7
8  sudo apt-get update
9
10 apt-cache policy docker-engine
11
12 sudo apt-get install -y docker-engine
13
14 sudo systemctl status docker

```

Fonte: Do autor (2018).

Na figura 27 foi demonstrado os comandos executados no servidor SRV001 para a instalação do Docker na versão 18.06.1-ce. Com o Docker devidamente

instalado no servidor é necessário a criação de dois contêineres, um container para o servidor de páginas web e outro para o servidor de banco de dados conforme o modelo proposto na Figura 26. O software utilizado para o servidor de página web será o apache na versão 2.4. Já para o servidor de banco de dados será utilizado a versão 5.7 do software MYSQL. Na figura 28 será demonstrando como e criar os dois contêineres descritos para este cenário.

Figura 28 - Lista de comando para instalação do Apache e MYSQL

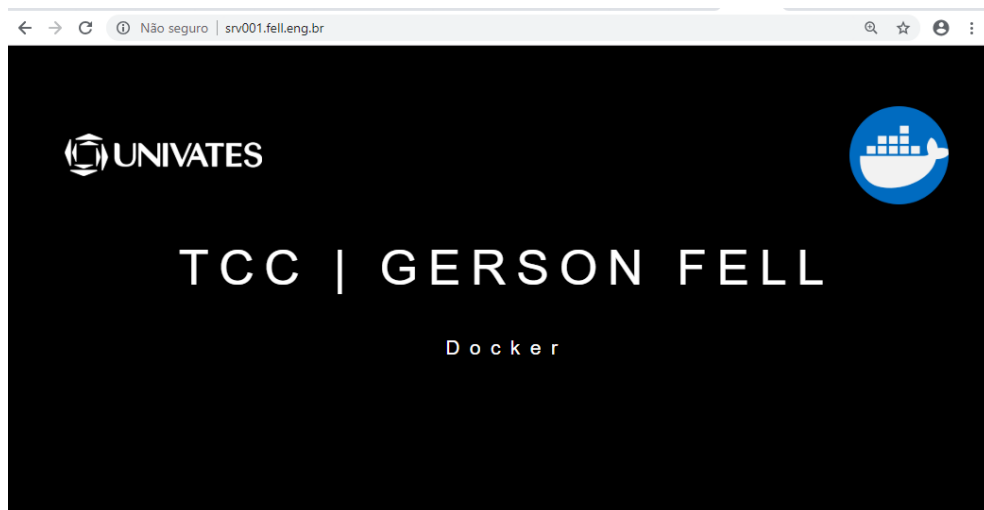
```
1 // COMANDO PARA BAIXAR ÚLTIMA VERSÃO DO APACHE
2 sudo docker pull httpd
3
4 // COMANDO PARA CRIAR O CONTAINER APACHE
5 sudo docker run -dit --name apache -p 80:80 -v /home/user/
  website:/usr/local/apache2/htdocs/ httpd:2.4
6
7 // COMANDO PARA INICIAR O CONTAINER APACHE
8 sudo docker start apache
9
10 // COMANDO PARA BAIXAR ULTIMA VERSÃO DO MYSQL
11 sudo docker pull mysql/mysql-server:latest
12
13 // COMANDO PARA CRIAR O CONTAINER MYSQL
14 sudo docker run --name mysql -e MYSQL_ROOT_HOST=% -e
  MYSQL_ROOT_PASSWORD=1234 -p 3307:3307 -d mysql/mysql-server
15
16 // COMANDO PARA INICIAR O CONTAINER MYSQL
17 sudo start mysql
```

Fonte: Do autor (2018).

Após a criação e inicialização dos contêineres é necessário configurar e testar os serviços Apache e MYSQL. Para o servidor de página web foi desenvolvido um layout de página específica para identificação do container Docker. Após a criação do layout é necessário efetuar a cópia destes arquivos para dentro do diretório */usr/local/apache2/htdocs/* que faz um link simbólico² para container Apache.

² Em computação, uma ligação simbólica é um tipo especial de arquivo que contém uma referência a outro arquivo ou diretório na forma de um caminho absoluto ou relativo e que afeta a resolução do nome de caminho

Figura 29 - Teste de acesso externo no container Apache do Docker SRV001.



Fonte: Do autor (2018).

A figura 29 foi demonstrado o sucesso no acesso da página web do servidor SRV001 que está utilizando a tecnologia de containerização. Já para o serviço MYSQL é necessário criar um usuário específico para o acesso externo (FIGURA 30).

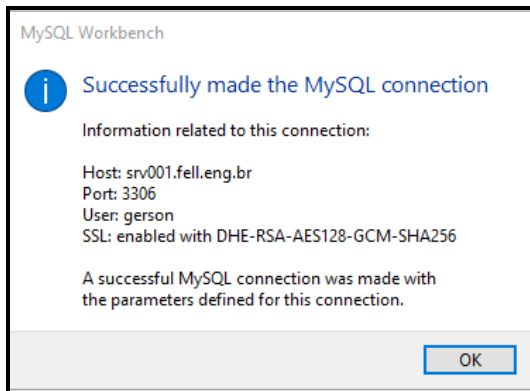
Figura 30 - Configuração de usuário para acesso externo MYSQL.

```
1 //ACESSAR LOCALMENTE O BANCO DE DADOS
2 sudo docker exec -it mysql mysql -p
3
4 // CRIAR UM USUARIO COM ACESSO EXTERNO
5 GRANT ALL ON *.* TO 'gerson'@'%' IDENTIFIED BY '29PQ71ij@@' WITH GRANT OPTION;
```

Fonte: Do autor (2018).

Os comandos demonstrados na figura 30 devem ser executados de dentro do container Docker MYSQL, pois o padrão do bando de dados MYSQL é bloquear todos os acessos externos. Após este procedimento finalizado é necessário efetuar o teste de acesso externo ao banco de dados MYSQL (FIGURA 31).

Figura 31 - Teste de acesso externo no container MYSQL do Docker SRV001.



Fonte: Do autor (2018).

Para efetuar o teste de acesso externo com o banco de dados foi utilizado a própria ferramenta nativa do banco de dados o MYSQL Workbench conforme demonstrado na figura 31.

5.1.2 Cenário 2

O cenário 2 foi implementado no SRV002. A figura 32 ilustra a configuração lógica do cenário 2. Para a virtualização do sistema operacional Ubuntu 18.04 foi utilizado o modelo de *hypervisor* do tipo 1.

Figura 32 - Estrutura lógica do cenário 2.



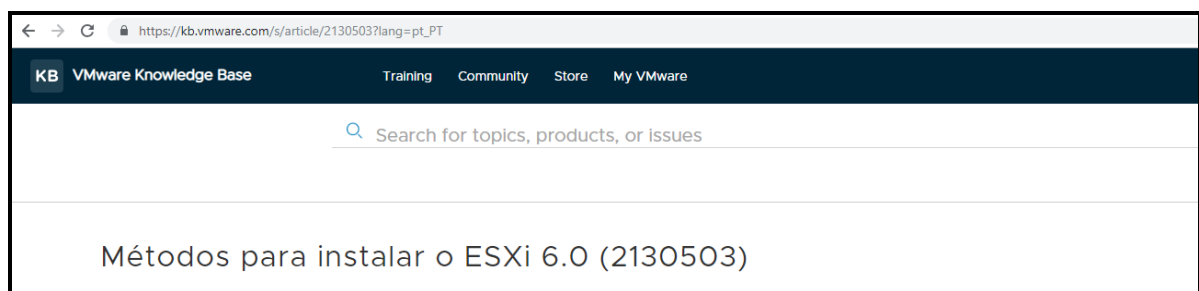
Fonte: Do autor (2018).

Conforme a figura 32 é possível verificar que neste cenário existe o mesmo número de camadas do cenário 1, porém neste cenário a diferença que foi efetuado

a instalação do sistema operacional sobre um *hypervisor* do tipo 1 e os serviços de página web e banco de dados estão sendo executados diretos no sistema operacional sem o uso da tecnologia de containerização.

A instalação do hypervisor e da máquina virtual foi efetuada seguindo o guia de instalação do software ESXi 6.0 que possui todo o passo a passo disponível no site da empresa conforme a figura 33.

Figura 33 - Tutorial para instalação do ESXi 6.0.



Fonte: Disponível em <https://kb.vmware.com/s/article/2130503?docid=2109708>.

Para fins de documentação do projeto a figura 34 demonstra as configurações definidas da máquina virtual criada no software de virtualização ESXi. A instalação do sistema operacional Ubuntu virtualizado segue os mesmos padrões do cenário 1 para assim garantir a integridade dos testes. A instalação dos serviços de servidor de página web e servidor de banco de dados sofrem algumas alterações do cenário anterior, pois neste cenário estes serviços devem ser instalados diretamente no sistema operacional sem a intervenção do Docker.

Figura 34 - Configuração da máquina virtual SRV002

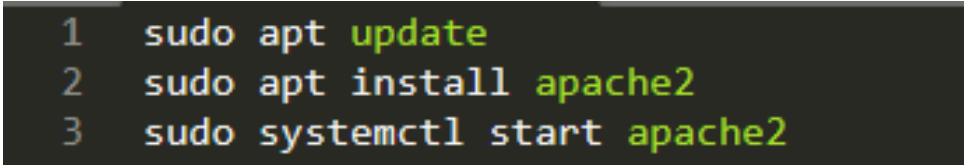
Hardware Configuration	
CPU	12 vCPUs
Memory	120.12 GB
Hard disk 1	80 GB

Fonte: Do autor (2018).

Na figura anterior foi demonstrado a configuração da máquina virtual criada sobre o hypervisor. O sistema ESXi permitiu a utilização de todos os recursos do hardware físico do servidor Dell na criação da máquina virtual, exceto a memória RAM, que teve uma diferença de 8GB quando comparado ao cenário 1.

Dando sequência ao projeto após a instalação do sistema operacional Ubuntu sobre a máquina virtual é necessário efetuar a instalação do serviço apache que necessita uma sequência de comandos que serão ilustrados na figura 35.

Figura 35 - Comando para instalação do Apache no SRV002.



```
1 sudo apt update
2 sudo apt install apache2
3 sudo systemctl start apache2
```

Fonte: Do autor (2018).

Após a inicialização do serviço de servidor de página web demonstrada na figura 35 é necessário copiar os arquivos do layout desenvolvido para identificação da máquina virtual para o diretório `/var/www/html/` do sistema operacional virtualizado Ubuntu 18.04. Após a cópia do layout para o diretório foi efetuado o teste de acesso externo no serviço de página da máquina virtual (FIGURA 36).

Figura 36 - Teste de acesso externo no Apache da máquina virtual SRV002.



Fonte: Do autor (2018).

A figura 36 foi demonstrado o sucesso no acesso da página web do servidor SRV002 que faz uso da tecnologia de virtualização para hospedar o sistema operacional, assim seguindo estrutura lógica do cenário 2 demonstrado na figura 32.

Após a instalação do serviço de página web é necessário efetuar a instalação do servidor de banco de dados que também é necessário a execução de alguns comandos que está ilustrado na figura 37. Nesta mesma figura foi efetuado a configuração do acesso externo.

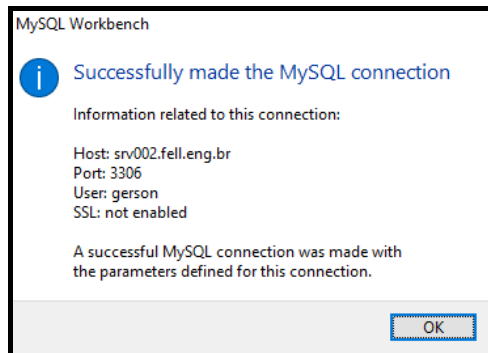
Figura 37 - Comando para instalação do MYSQL no SRV002.

```
1 //Instalação do MYSQL
2 sudo apt update
3 sudo apt install mysql-server
4 // Configuração do acesso externo
5 mysql -uroot -p
6 mysql> GRANT ALL ON *.* TO 'gerson'@'%' IDENTIFIED BY '29PQ71ij@@' WITH GRANT OPTION;
```

Fonte: Do autor (2018).

Como podemos ver na figura 37 foi criado um usuário específico para obter o acesso externo do servidor de banco de dados que por padrão vem bloqueado. Para efetuar o teste de acesso externo foi utilizado o mesmo software do cenário 1 (FIGURA 38).

Figura 38 - Teste de acesso externo no MYSQL máquina virtual SRV002.



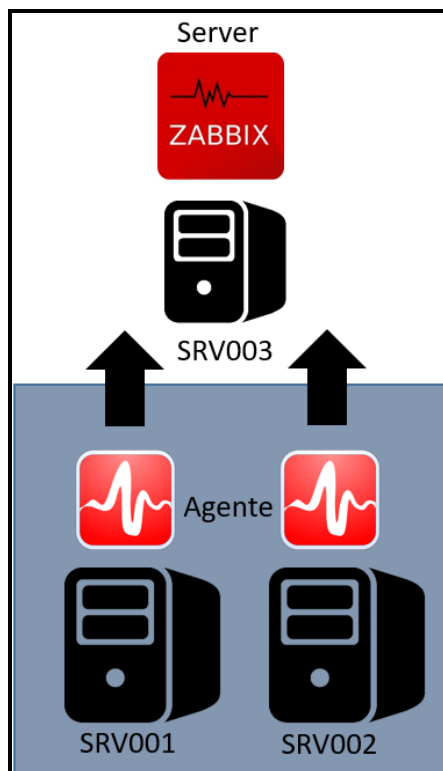
Fonte: Do autor (2018).

Conforme demonstrado na figura 38 foi obtido sucesso ao acessar o servidor de banco de dados que está sendo executado sobre a máquina virtual SRV002.

5.2 Coleta de dados

Para a coleta dos dados foi utilizado o software Zabbix³. Este software será instalado no servidor SRV003 conforme a figura 39.

Figura 39 – Cenário de configuração do software Zabbix.



Fonte: Do autor (2018).

³ O Zabbix é uma ferramenta de software de monitoramento de código aberto para diversos componentes de TI, incluindo redes, servidores, máquinas virtuais e serviços em nuvem.

Na figura 39 foi ilustrado a configuração lógica do software Zabbix para a coleta de dados. Para conseguir efetuar esta coleta é necessário instalar dois serviços sendo no SRV003 o serviço Zabbix server e nos servidores de testes SRV001 e SRV002 o Zabbix agente. As instalações dos serviços foram feitas seguindo o artigo de Instalação do Zabbix 3.x para Ubuntu disponibilizado pelo Autor Aécio Pires (2016). Seguindo as orientações de instalação e configuração do artigo foi obtido sucesso que pode ser verificado na figura 40.

Figura 40 – Cenário de configuração do software Zabbix.

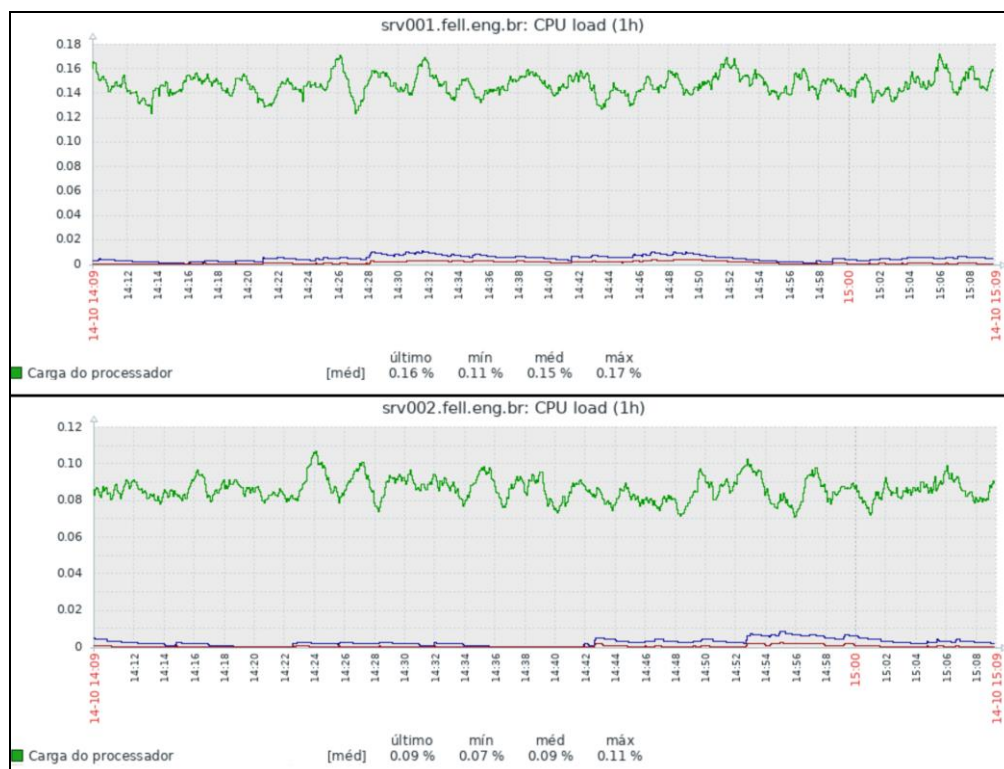
<input type="checkbox"/>	Nome ▲	Aplicações	Itens	Triggers	Gráficos	Descoberta	Web	Interface	Templates	Status	Disponibilidade
<input type="checkbox"/>	srv001.fell.eng.br	Aplicações 10	Itens 73	Triggers 17	Gráficos 19	Descoberta 2	Web 177.44.248.125:10050		Template OS Linux (Template App Zabbix Agent)	Ativo	ZBX SNMP JMX IPMI
<input type="checkbox"/>	srv002.fell.eng.br	Aplicações 10	Itens 55	Triggers 17	Gráficos 10	Descoberta 2	Web 177.44.248.126:10050		Template OS Linux (Template App Zabbix Agent)	Ativo	ZBX SNMP JMX IPMI

Fonte: Do autor (2018).

Como pode ser verificado na figura 40 a comunicação entre o serviço Zabbix server e Zabbix agente estão funcionando. O software Zabbix utilizada um protocolo próprio para efetuar a coleta de dados que na imagem acima está ilustrado com o status ativo nos servidores SRV001 e SRV002.

Segundo Aécio Pires (2016) um dos recursos que dá mais agilidade ao Zabbix é o recurso de *templates*. Seguindo as orientações do autor foi feito o uso de *templates* para coletas de dados, assim possibilitando a visualização dos primeiros gráficos de consumo de recursos computacionais (FIGURA 41).

Figura 41 – Consumo da CPU dos servidores com uso de *templates*.



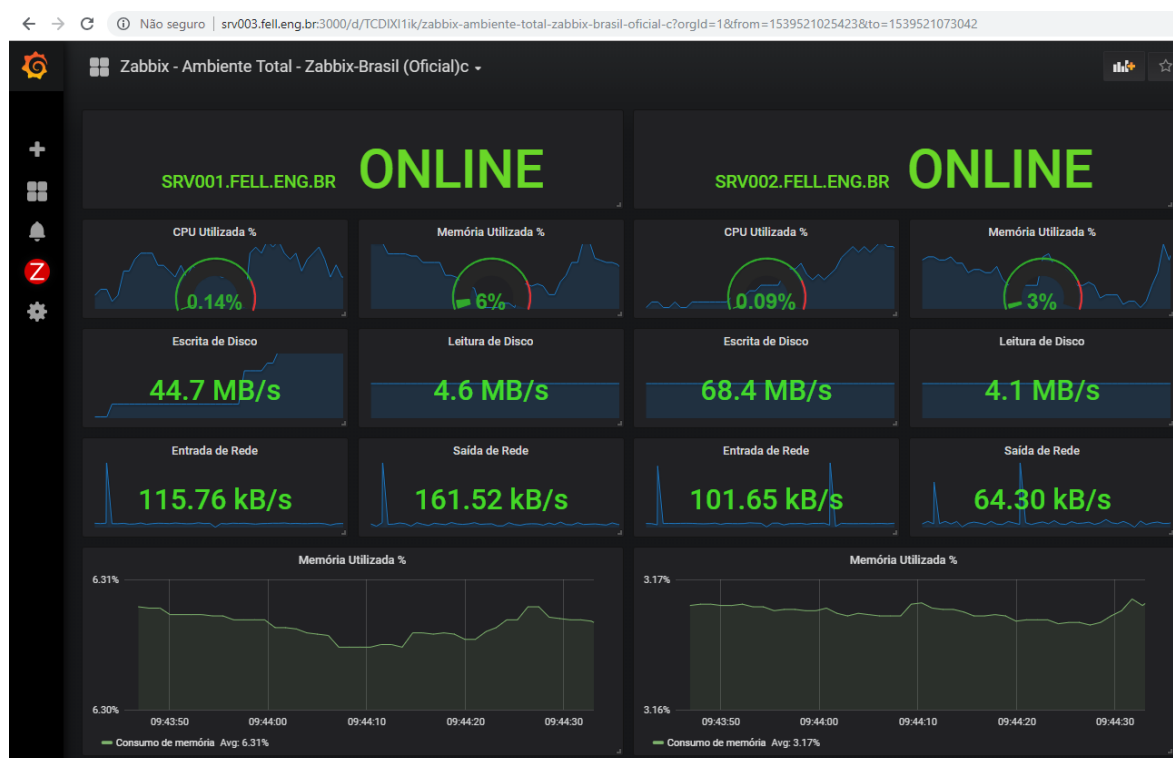
Fonte: Do autor (2018).

Conforme ilustrado na figura 41 foi obtido sucesso na coleta dos dados nos servidores de testes, porém foi verificado que o software Zabbix não permitia o compartilhamento das telas e gráficos em tempo real. Como o escopo do projeto previa o desenvolvimento uma aplicação para efetuar uma análise de despenho das tecnologias de containerização e virtualização tradicional em tempo real, foi necessário executar uma pesquisa sobre bibliotecas para o Zabbix ou até mesmo outros softwares para efetuar a coleta e também o compartilhamento de gráficos e tabelas em tempo real, sem ter necessidade de acessar o próprio painel de administração do software Zabbix.

A pesquisa por outros softwares obteve-se sucesso, pois existe hoje no mercado uma ferramenta de software livre que tem como seu principal intuito o compartilhamento em tempo real dos dados do software Zabbix. Segundo o autor Jorge Pretel (2017) a ferramenta denominada de Grafana tem a finalidade de utilizar os dados coletados no Zabbix para uma visualização com flexibilidade de gráficos e outras funcionalidades que não são possíveis utilizando apenas o Zabbix. Para realizar esta integração o autor escreveu um artigo chamado de Integração do

Zabbix com Grafana. Neste artigo contém toda a orientação necessária para a instalação e também a confecção de gráficos com a ferramenta Grafana. Após a instalação e configuração foi possível ter acesso aos primeiros gráficos utilizando o Grafana conforme ilustra a figura 42.

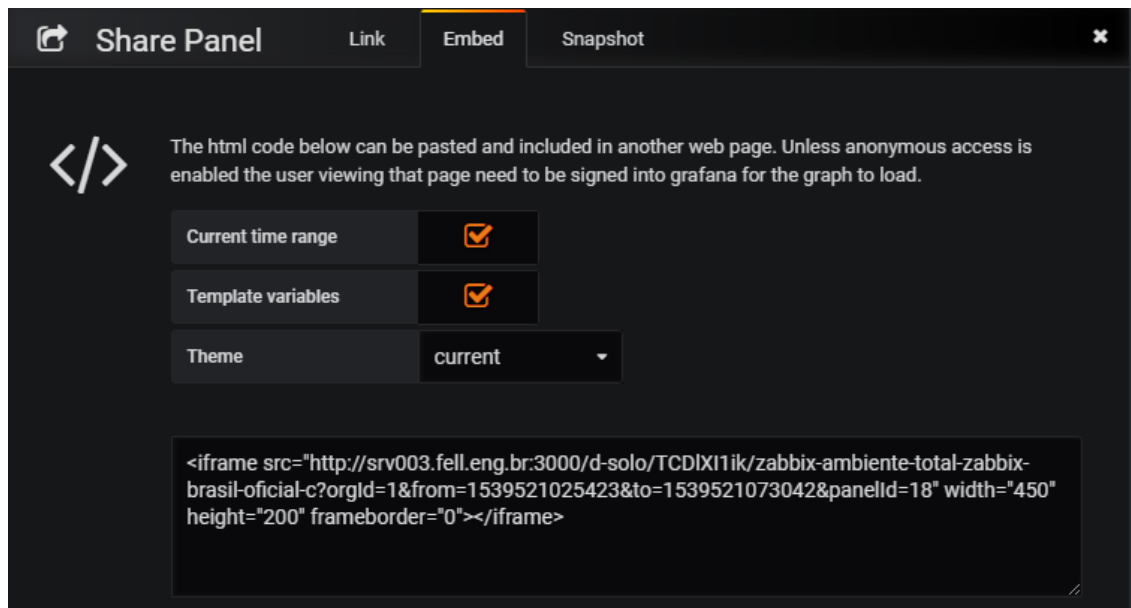
Figura 42 – Página de administração do Grafana utilizando o Zabbix.



Fonte: Do autor (2018).

É possível analisar na figura 42 que os gráficos do Grafana têm um layout mais elaborado quando comparados os da ferramenta Zabbix como demonstrado na figura 41. O autor Jorge Pretel também faz uma rápida demonstração de como compartilhar os gráficos do Grafana utilizando *iframes*, que segundo ele são quadros que podem ser adicionados facilmente em páginas HTML, exibindo assim as informações do Grafana em um site ou em um aplicativo (FIGURA 43).

Figura 43 – Compartilhamento de gráficos utilizando o Grafana.



Fonte: Do autor (2018).

Conforme ilustrado na figura 43 o *software* Grafana atendeu as necessidades de compartilhando de telas e gráficos em tempo real, com isso o próximo passo será desenvolver uma aplicação para efetuar rajadas de acessos nos serviços de páginas e rajadas de *consulta e inserção de dados* nos serviços de bancos de dados dos servidores SRV001 e SRV002.

5.3 Aplicação mestre

Esta aplicação terá o intuito de gerar rajadas em ambos os servidores e também exibir o consumo dos recursos computacionais dos servidores em tempo real. Para isso se tornar possível foi desenvolvido uma aplicação web utilizando a linguagem Java Server Pages (JSP) que é uma tecnologia que auxilia no desenvolvimento de páginas web dinâmicas baseando se em HTML e também para agilizar e facilitar a criação destas páginas dinâmicas foi utilizado um framework. O framework escolhido para foi o ZK que visa combinar a simplicidade e a segurança de sua arquitetura centrada no lado do servidor. Está aplicação será hospedada no servidor SRV003 que fara o uso do serviço de páginas Tomcat 7, com isso será necessário realizar a instalação desta ferramenta executando os comandos ilustrados na figura 44.

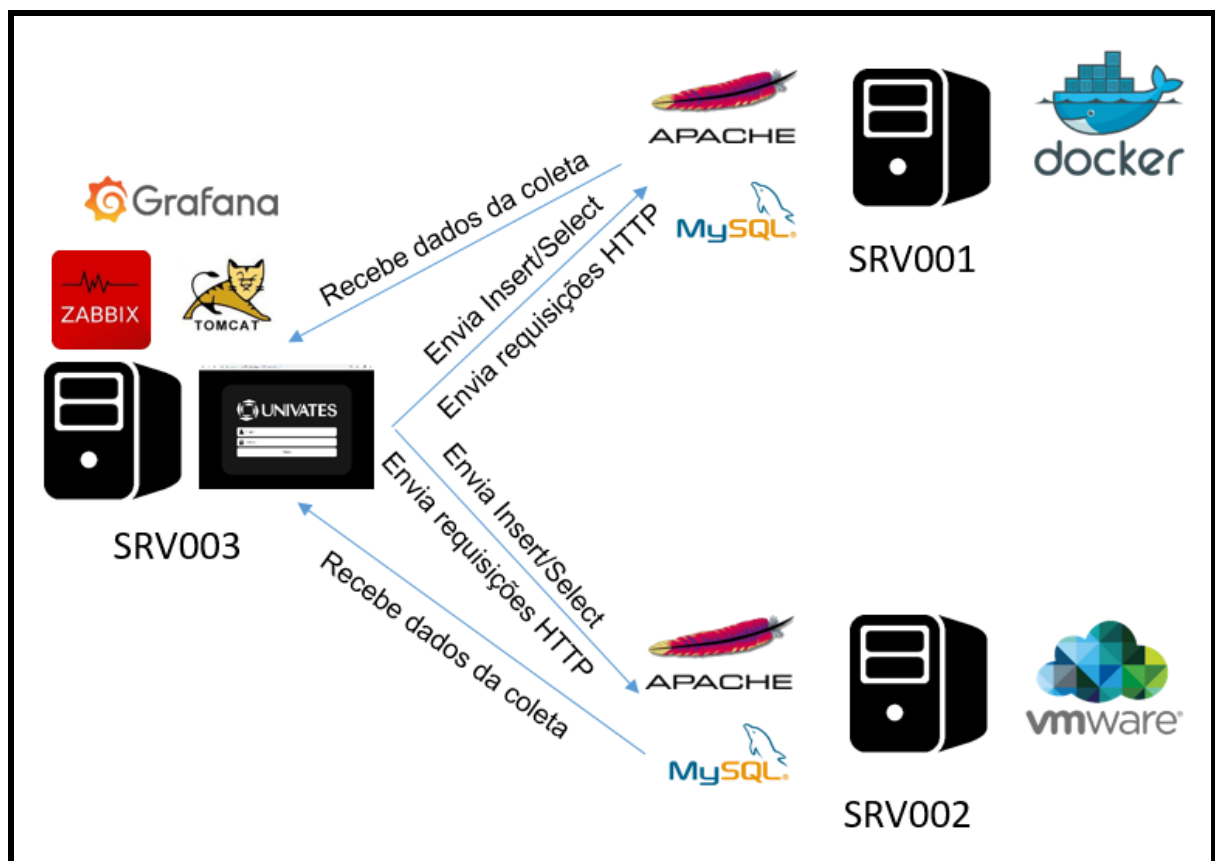
Figura 44 – Instalação do servidor de página Tomcat 7.

```
1 sudo apt-get update
2 sudo apt-get install tomcat7
3 sudo service tomcat7 restart
```

Fonte: Do autor (2018).

Na Figura 45, é apresentado um diagrama macro da visão do projeto da aplicação, nesta figura é apresentado a arquitetura do projeto e sua operabilidade.

Figura 45 – Diagrama da aplicação central.



Fonte: Do autor (2018).

Como é possível verificar na figura 45 o SRV003 centraliza a função de gerador de rajadas para os testes e a função de coleta de dados. Os testes descritos na figura serão explicados nas seções 5.3.1 e 5.3.2. Já a coletas de dados foi explicada na seção 5.2.

5.3.1 Teste MYSQL

Para este modelo de teste foi desenvolvido um código de inserção e seleção de dados. O algoritmo tem uma ordem de execução nos testes, ou seja, ele vai efetuar uma inserção e após finalizar vai efetuar uma consulta.

Na figura 46 é ilustrado a ordem que é executado os comandos na base de dados. Nesta figura é possível visualizar que é o código é executado em uma tabela nomeada de per_1, porém existe outra tabela igual nomeada de per_2 que tem a mesma função e colunas. Os campos destas tabelas também são iguais sendo que as primeiras cinco colunas serão gravados *hash* randomicamente e os ultimas cinco colunas serão gravadas imagens no formato binário.

Figura 46 – Código de inserção e seleção na base de dados.

```
String sql = "insert into per_1 values ( MD5(RAND()), MD5(RAND()), MD5(RAND()), MD5(RAND()), MD5(RAND()), ?, ?, ?, ?, ?)";
LogUtilities.getInstance().write( sql );

PreparedStatement ps = db.getPreparedStatement( sql );

ps.setBinaryStream( 1, ImageProvider.IMG.get( 1 ) );
ps.setBinaryStream( 2, ImageProvider.IMG.get( 2 ) );
ps.setBinaryStream( 3, ImageProvider.IMG.get( 3 ) );
ps.setBinaryStream( 4, ImageProvider.IMG.get( 4 ) );
ps.setBinaryStream( 5, ImageProvider.IMG.get( 5 ) );

try
{
    ps.execute();
}

finally
{
    ps.close();
}

sql = "select * from per_1";
```

Fonte: Do autor (2018).

Este código demonstrado na figura 46 espera dois parâmetros sendo eles o número de usuários e a quantidade de vezes que cada usuário vai inserir e selecionar as tabelas per_1 e per_2. Na figura 47 será demonstrado o trecho do código que espera o parâmetro do número de usuários. Para isto é criado *threads* para cada usuário adicionado possibilitando a simulação de múltiplos usuários inserindo e selecionado ao mesmo tempo, simulando uma concorrência de acessos nos servidores de testes.

Figura 47 – Código simulando a criação múltiplos usuários.

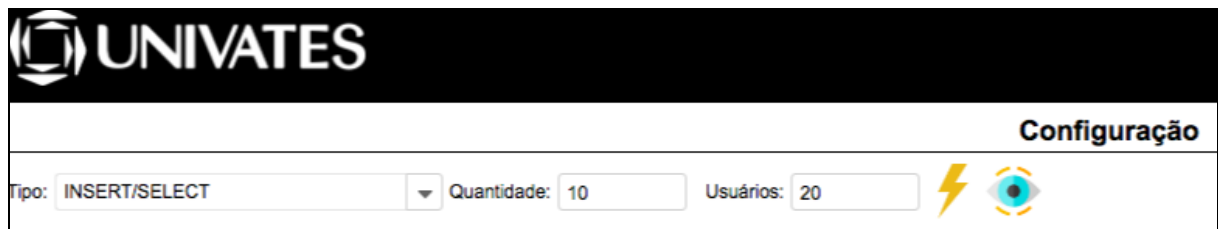
```
private void doStatment( Event evt ) {
    StatmentData data = statmentPane.getData();
    for ( int i = 0; i <= data.getUser(); i++ ) {
        fireStatment( i, Base.DOCKER );
        fireStatment( i, Base.VIRTUAL_MACHINE ); }
    Prompts.info( "Executando..." );
}
private void fireStatment( int user, Base base ) {
    StatmentData data = statmentPane.getData();
    Thread t = new Thread(() ->
    { try {
        Database db = Database.getInstance( base );
        LogUtilities.getInstance().write( "Conectando ao " + base + "...." );
        try {
            switch ( data.getType() ) {
                case MIXED:
                    StatmentController.doStatment( data, db );
                    break;
                case HTTP:
                    RequestController.doRequest( data, base );
                    break; } }
            finally
            { LogUtilities.getInstance().write( "Disconectando ao " + base + "...." );
              db.release();
            } }
        catch ( Exception e )
        {
            e.printStackTrace( System.err );
        } } );
    t.setName( base.toString() + " - " + ( user + 1 ) );
    t.setDaemon( true );
    t.start(); }
```

Fonte: Do autor (2018).

Na figura 47 também é possível analisar que é entregue um objeto nomeado de data para chamar a classe *controller*. Esse objeto contém o atributo quantidade que se refere ao número de repetição que cada usuário irá executar as operações no banco de dados. Para melhor entendimento na aplicação foi chamada de ciclo esta repetição, ou seja, cada vez que for iniciada uma *thread* simulando o usuário e aberto um ciclo e quando finalizada encerra o ciclo. Para entender melhor este conceito é necessário analisar a figura 49 que será demonstrado os logs do teste da aplicação mestre.

Dando sequência foi desenvolvido uma página web que será ilustrado na figura 48, esta página web demonstra um teste de inserção e seleção em andamento utilizando os parâmetros número de usuários e quantidade de repetições.

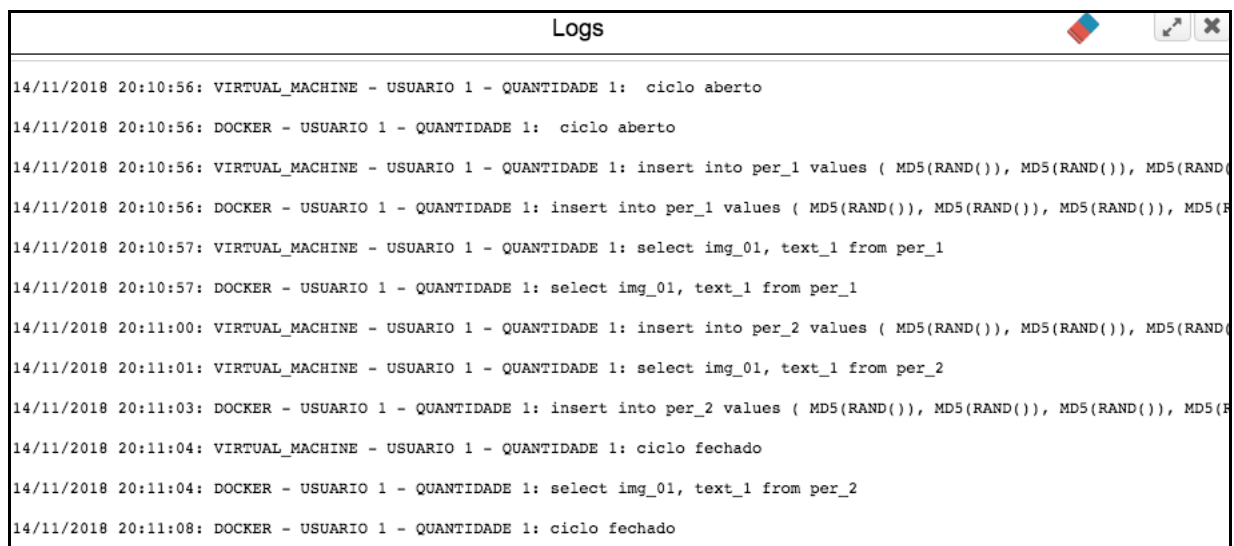
Figura 48 – Aplicação mestre efetuando testes de inserção e seleção.



Fonte: Do autor (2018).

A figura 48 ilustra um teste sendo executado com o parâmetro de número de usuário definido em vinte e o parâmetro de quantidade em 10. Para conseguir efetuar uma análise do que está ocorrendo durante os testes de inserção e seleção foi desenvolvido um recurso na aplicação que possibilita a visualização dos comandos executados nas bases de dados dos servidores demonstrada na figura 49.

Figura 49 – Recurso de análise de logs do teste de inserção e seleção.



Fonte: Do autor (2018).

Conforme ilustrado na figura 49 os comandos são executados de uma forma paralela entre os servidores de testes, sempre seguindo a ordem de inserir e depois consultar a tabela.

5.3.2 Teste Apache

Neste modelo de teste foi utilizado uma ferramenta desenvolvida pela própria Apache para testar e medir o desempenho de qualquer servidor de página web. O

Apache Bench é um *software* que é executado por linha de comando que aceita três parâmetros, sendo eles os parâmetros número de páginas, número de usuários e por fim o nome do servidor que será gerado os testes (FIGURA 50).

Figura 50 – Execução do Apache Bench no servidor SRV003.

```

gerson@srv003:~$ ab -n 90 -c 80 -k srv005.fell.eng.br/
This is ApacheBench, Version 2.3 <Revision: 1807734 >
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking srv005.fell.eng.br (be patient).....done


Server Software:      Apache/2.4.35
Server Hostname:      srv005.fell.eng.br
Server Port:          80

Document Path:        /
Document Length:      1883 bytes

Concurrency Level:    80
Time taken for tests:  0.009 seconds
Complete requests:    90
Failed requests:       0
Keep-Alive requests:  90
Total transferred:    195020 bytes
HTML transferred:     169470 bytes
Requests per second:  10252.90 [#/sec] (mean)
Time per request:      7.803 [ms] (mean)
Time per request:      0.098 [ms] (mean, across all concurrent requests)
Transfer rate:         21696.20 [Kbytes/sec] received

Connection Times (ms)
              min    mean[+/-sd] median    max
Connect:        0      3   1.1      3      4
Processing:      1      2   0.7      2      4
Waiting:         1      2   0.7      2      4
Total:           1      5   1.5      5      8

Percentage of the requests served within a certain time (ms)
 50%    5
 66%    6
 75%    6
 80%    6
 90%    6
 95%    7
 98%    8
 99%    8
100%    8 (longest request)

```

Fonte: Do autor (2018).

Conforme ilustrado na figura 50 foi efetuado um teste com 80 usuarios e cada um destes usuarios simulou o acesso em 90 páginas no servidor SRV002. Também é possível visualizar que o Apache Bench retorna diversas informações referente ao teste executado. Após obtido sucesso utilizando o Apache Bench direto por linha de comando no servidor SRV003 foi necessário desenvolver uma função que executa esta linha de comando por dentro da aplicação mestre, para assim não ser necessário acessar o servidor por linha de comando toda vez que for executar o teste de página web (FIGURA 51).

Figura 51 – Função para executar o Apache Bench na aplicação mestre.

```
public static void doRequest( StatmentData data, Base base )
{
    try {
        LogUtilities.getInstance().write( -1, "Requisitando " + base.host() + ":" + base.port() );
        ProcessBuilder builder = new ProcessBuilder( "ab", "-n", String.valueOf( data.getPaginas() * data.getUser() ) +
            "-c", String.valueOf( "50" ), "-k", base.host() + ":" + base.port() );
        builder.redirectErrorStream( true );
        BufferedReader input = new BufferedReader(new InputStreamReader(builder.start().getInputStream()));
        String line, lines = "\n";
        try {
            while ((line = input.readLine()) != null) {
                lines += line + System.lineSeparator();
            }
            LogUtilities.getInstance().write( -1, lines );
        } catch (IOException e) {
            LogUtilities.getInstance().write( -1, "ERROR: " + e.getMessage() );
        }
    }
    catch ( Exception e ) {
        try {
            LogUtilities.getInstance().write( -1, "ERROR: " + e.getMessage() );
        } catch ( Exception ex ) {
            ApplicationContext.getInstance().logException( ex );
        }
    }
}
```

Fonte: Do autor (2018).

Conforme demonstrado na figura 51 foi desenvolvido uma função que inicia o software Apache Bench por linha de comando no servidor SRV003, assim possibilitando executar o teste nos servidores SRV001 e SRV002 direto pela aplicação mestre. Após este passo concluído foi necessário desenvolver um *layout* web para o teste em questão (FIGURA 52).

Figura 52 – Aplicação mestre configurada para efetuar teste de requisição.

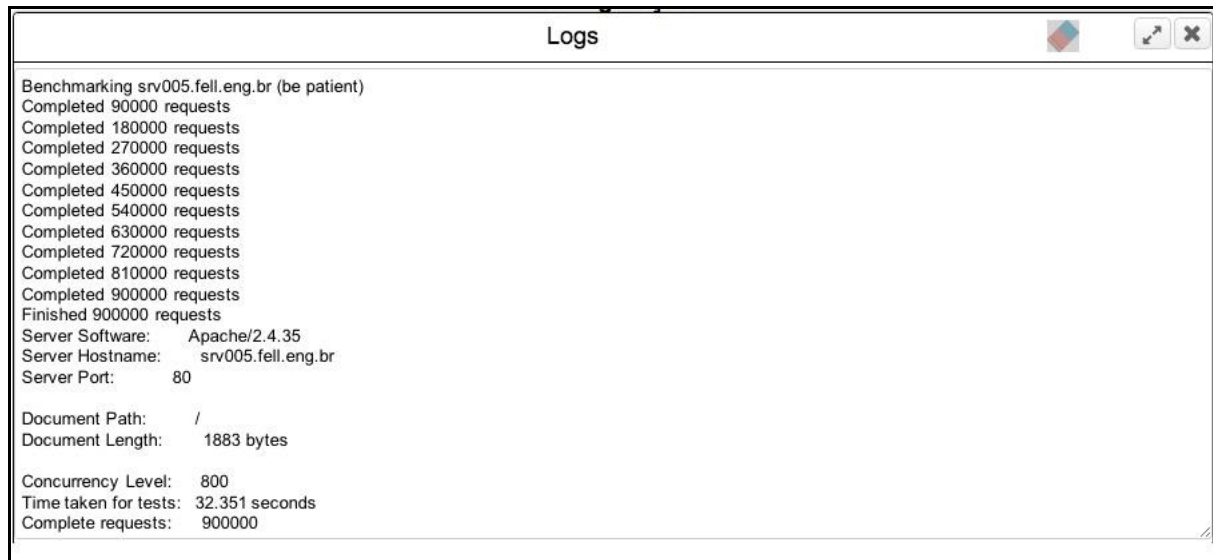


Fonte: Do autor (2018).

Conforme ilustrado na figura 52, a aplicação mestre tem dois campos disponíveis para configuração dos testes de páginas web. Nesta figura foi efetuado uma simulação de um teste com dez usuários acessando simultaneamente o serviço de página web dos servidores SRV001 e SRV002, sendo que cada usuário vai efetua três requisições em ambos os servidores.

Seguindo o mesmo padrão dos testes de inserção e seleção foi desenvolvido um recurso na aplicação mestre que possibilita efetuar uma análise mais completa das requisições que estão sendo feita nos servidores de teste (FIGURA 53).

Figura 53 – Recurso de análise de logs do teste de requisição.



Fonte: Do autor (2018).

Conforme demonstrado na figura 53 é possível ter uma análise mais robusta de como o software Apache Bench sendo executado e por fim retornando uma análise completa de toda a sua execução.

5.4 Configurações gerais

Como padronização das configurações de softwares utilizados no projeto foi desenvolvido uma ilustração com todos os parâmetros que foram necessários ser alterados, sendo assim no quadro 01 será apresentado os parâmetros que não seguiram com as configurações padrão dos softwares utilizados (QUADRO 01).

Quadro 1 - Configurações gerais dos softwares utilizados.

	<p>A coleta de dados será efetuada a cada segundo nos servidores SRV001 e SRV002</p>
	<p>Os gráficos serão atualizados a cada segundo. Será exibido nestes gráficos apenas os últimos 5 minutos coletados nos servidores SRV001 e SRV002</p>
	<p>A memória máxima configurado para o Tomcat será de 100GB no SRV003</p>
	<p>O número máximo de conexões simultâneas dos servidores de banco de dados foi definido como 10,000 acessos nos servidores SRV001 e SRV002.</p>

Fonte: Do autor (2018).

Conforme demonstrado na figura 38 foram alterados quatro parâmetros de configuração dos softwares utilizados. Os parâmetros alterados do Zabbix e Grafana foi necessário, pois houve algumas dificuldades na coleta de dados e também na geração dos gráficos já o Tomcat e MYSQL foi necessário para conseguir-se obter um melhor desempenho nos testes aplicados.

6 TESTES E ANÁLISES

Neste capítulo será apresentado os testes executados nos cenários propostos no capítulo 5 e consecutivamente será efetuado uma análise sobre os resultados obtidos. Para um melhor entendimento cada teste será dividido em um subcapítulo e dentro deste subcapítulo será demonstrado os gráficos do consumo de memória, consumo de CPU, I/O de disco e *throughput* de rede durante cada teste efetuado. O padrão dos gráficos que serão exibidos nas próximas seções será utilizado como padrão a seguinte organização:

- Gráfico posicionado ao lado esquerdo refere-se ao resultado referente ao servidor SRV001 que faz uso da tecnologia de containerização;
- Gráfico posicionado ao lado direito refere-se ao resultado referente ao servidor SRV002 que faz o uso da tecnologia de máquina virtual do tipo 1.

Por fim será criado um subcapítulo para descrever as observações gerais observadas após os testes em questão.

6.1 Teste sem carga

O teste sem carga se refere a coleta de dados dos servidores em seu estado ocioso, ou seja, o consumo computacional que será apresentado nos gráficos a seguir são o real consumo que o sistema operacional utiliza para deixar em execução o serviço de banco de dados e de página web de ambos os servidores sem nem uma requisição sendo efetuada (GRÁFICO 1).

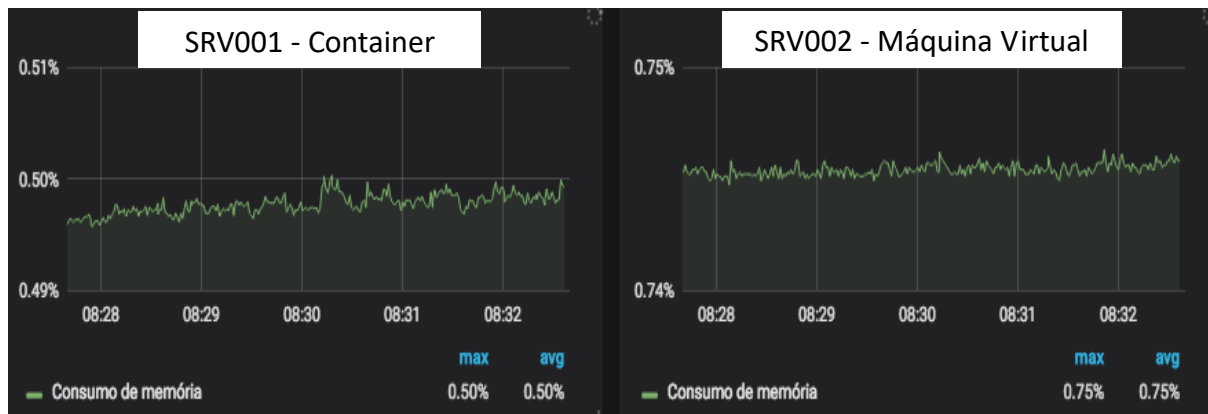
Gráfico 1 – Consumo de memória sem carga.



Fonte: Do autor (2018).

O gráfico 1 demonstra o real consumo de memória dos serviços de bancos e páginas web. Como podemos ver o consumo de memória entre as duas tecnologias é insignificante, pois tem uma diferença de 0,14% em uma média de 5 minutos de coleta. No consumo de CPU basicamente se obteve o mesmo resultado conforme o gráfico 2.

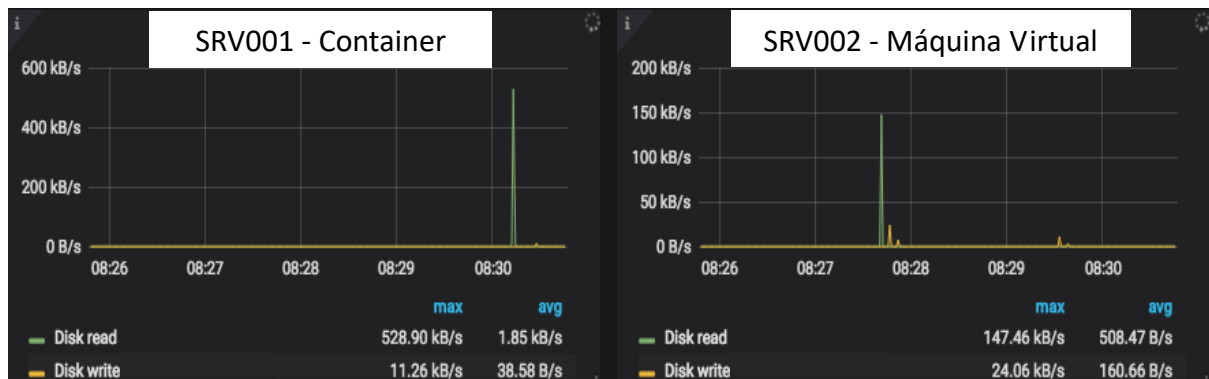
Gráfico 2 – Consumo de CPU sem carga.



Fonte: Do autor (2018).

No Gráfico 2 a diferença entre as duas tecnologias no quesito CPU foi baixa, pois teve uma média de 0,15% de diferença em uma coleta de 5 minutos. Os dados exibidos no gráfico 3 para analisar a leitura e escrita do disco rígido os dados foram coletados em KB/s e MB/s.

Gráfico 3 – Leitura e escrita de disco rígido sem carga.

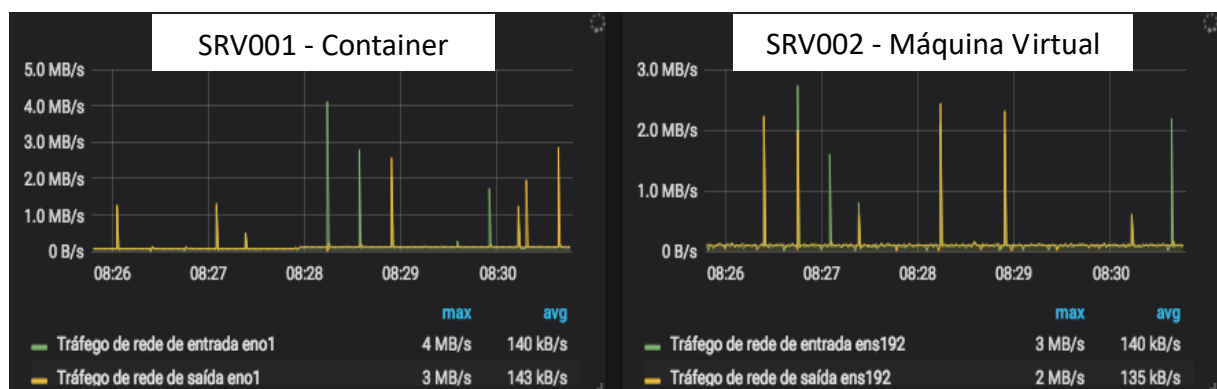


Fonte: Do autor (2018).

Conforme demonstrado no gráfico 3, também não houve uma grande diferença no consumo quando comparado a leitura e escrita do disco rígido entre as duas tecnologias.

O gráfico 4, vai utilizar o termo *throughput* para definir a leitura e escrita de dados no disco. O *throughput* pode ser traduzido como a taxa de transferência efetiva, ou seja, quantidade de dados transferidos de um lugar a outro, ou a quantidade de dados processados em um determinado espaço de tempo. Pode-se usar o termo *throughput* para referir-se à quantidade de dados transferidos em discos rígidos ou em uma rede.

Gráfico 4 – Entrada e saída de rede sem carga.



Fonte: Do autor (2018).

O gráfico 4 demonstra que também não existe uma diferença significativa de *throughput* de rede entre as tecnologias.

Efetuada uma análise geral em todos os gráficos demonstrados neste teste, o resultado já era previsto, pois quando é efetuado uma coleta do consumo de recursos computacionais com servidores ociosos, eles tendem a ser nulos. Isto ocorre pois só vai existir alguma alteração nestes dados quando houver serviços enviando ou recebendo informações para assim ser processadas e desta forma então consumir estes recursos computacional.

6.2 Testes com carga do MYSQL

Dando início ao teste, é necessário escolher os parâmetros de configuração para assim efetuar os testes com carga no serviço de banco de dados dos servidores SRV001 e SRV002. Estes parâmetros foram escolhidos após inúmeras simulações, pois o objetivo das simulações era encontrar uma configuração específica que atingisse em torno de 5 minutos de processamento nos servidores de testes, assim demonstrando como cada tecnologia se comporta durante este tempo. Na figura 54 será demonstrado os parâmetros escolhidos para a primeira etapa dos testes com carga.

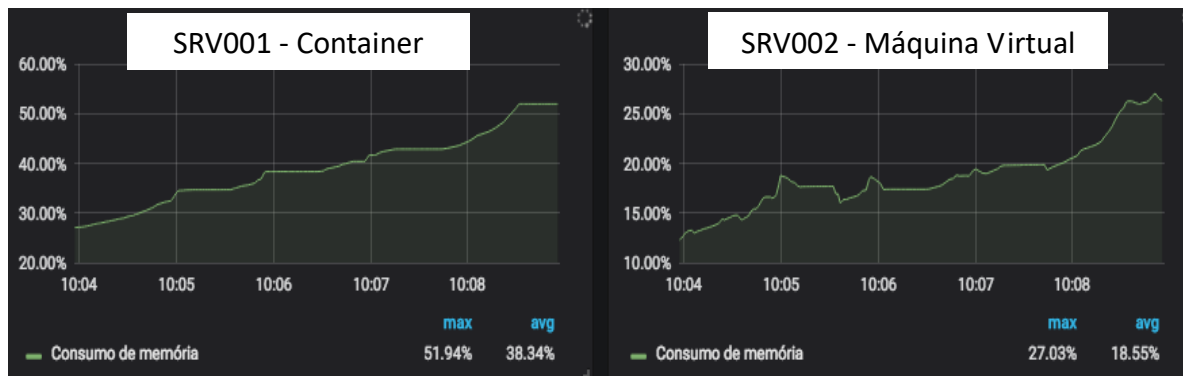
Figura 54 - Parâmetros utilizados no teste com carga do MYSQL.

The image shows a web interface for UNIVATES. At the top is the UNIVATES logo. Below it is a section titled "Configuração". Inside this section, there are three input fields: "Tipo:" with a dropdown menu showing "INSERT/SELECT", "Quantidade:" with a text box containing "100", and "Usuários:" with a text box containing "2200". To the right of these fields are four icons: a yellow lightning bolt, a blue lightning bolt with a database symbol, a yellow lightning bolt with a document symbol, and a blue eye icon.

Fonte: Do autor (2018).

Na figura 54 é ilustrado os parâmetros de configuração para geração do teste em questão. O teste com carga do MYSQL será configurado com 2200 usuários e cada usuário vai efetuar 100 repetições. O primeiro gráfico demonstrado será referente ao consumo de memória (GRÁFICO 5).

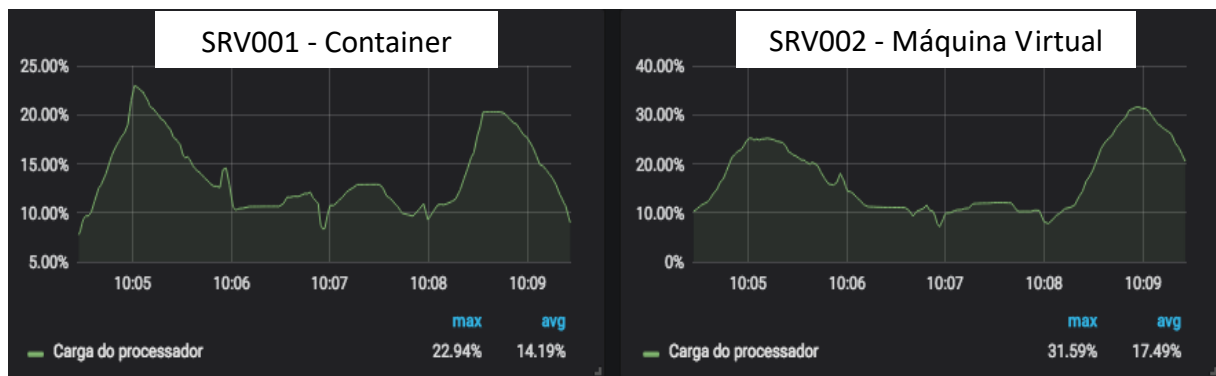
Gráfico 5 – Consumo de memória no teste com carga do MYSQL.



Fonte: Do autor (2018).

Como é possível analisar no gráfico 5, as duas tecnologias se comportaram de uma maneira diferente no aspecto consumo de memória, pois, o servidor SRV002 tem um consumo baixo de memória. Este consumo chega a ter uma diferença na média de 19,79% entre as tecnologias, lembrando ainda que o servidor SRV002 tem 10GB a menos de memória quando comparado o SRV001. No gráfico 6 será demonstrado o consumo de CPU dos servidores de testes.

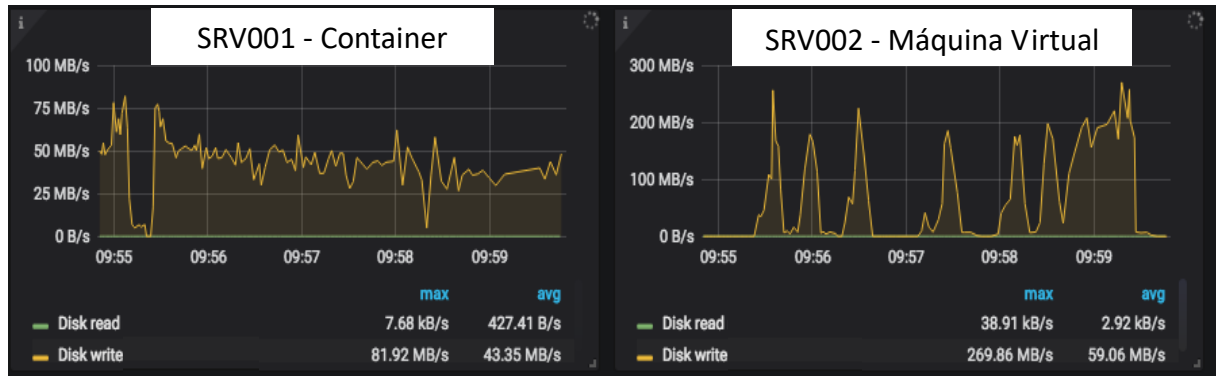
Gráfico 6 – Consumo de CPU no teste com carga do MYSQL.



Fonte: Do autor (2018)

O gráfico 6 não segue o mesmo padrão do gráfico de consumo de memória, pois ele tem um comportamento similar entre os servidores SRV001 e SRV002, tendo apenas uma diferença 3,3% na média entre os 5 minutos de coletas. Dando sequência, no gráfico 7 será demonstrado o I/O de disco rígido no teste com carga do MYSQL.

Gráfico 7 – Leitura e escrita de disco rígido no teste com carga do MYSQL.

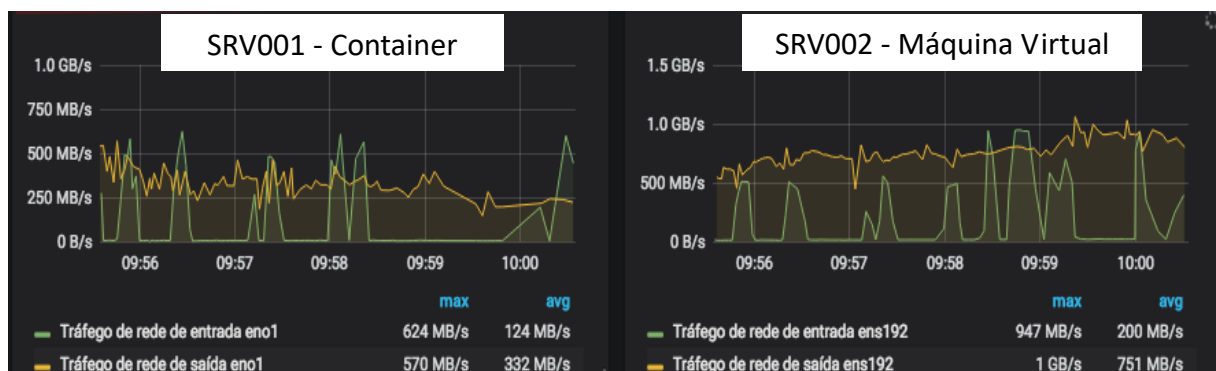


Fonte: Do autor (2018)

Como demonstrado no gráfico 7 a diferença entre as duas tecnologias foi de 187,94 MB/S quando se trata do pico máximo de gravação e uma diferença na média de 15,71 MB/s entre os servidores. Lembrando que está sendo suposto que os discos rígidos utilizados em ambos os servidores estão com mesmo estado de físico, ou seja, com a mesma rotação em ambos os servidores.

Também é importante analisar que a leitura do disco está extremamente baixa. Isso provavelmente acontece, pois, como está sendo efetuado sempre a mesma consulta na base de dados o banco MYSQL adiciona em memória o resultado desta consulta, assim não sendo necessário sempre efetuar a mesma busca no disco rígido, com isso causando o aumentando da leitura. Dando sequência no próximo gráfico será demonstrado a entrada e saída da placa de rede durante o teste com do serviço de banco de dados (GRÁFICO 8).

Gráfico 8 – Entrada e saída de rede no teste com carga do MYSQL.



Fonte: Do autor (2018)

No gráfico 8 é possível verificar que quando se trata de *throughput* de rede também existe uma real diferença entre as duas tecnologias, pois na média entre 5 minutos de coletas teve uma variação de 419 MB/s na saída e 76 MB/s na entrada de dados, lembrando que o limite físico da placa de rede é de 1000 MB/s tanto de saída quanto de entrada. Também é interessante analisar que existe uma diferença no pico máximo do *throughput* entre as tecnologias, esta diferença chega a ser de 430 MB/s na saída e de 323 MB/s na entrada de dados.

6.3 Testes com carga do Apache

Após a finalização dos testes com o serviço MYSQL, iniciamos os testes no Apache. Conforme demonstrado na figura 52 é necessário definir dois parâmetros, estes parâmetros foram definidos a partir de diversas simulações, porém nestas simulações não se conseguiu uma configuração exata para deixar o teste em execução durante os 5 minutos de coleta, pois o servidor SRV003 não tem uma capacidade física suficiente. Por este motivo foi definido a configuração ilustrada na figura 55.

Figura 55 - Parâmetros utilizados no teste com carga do Apache.

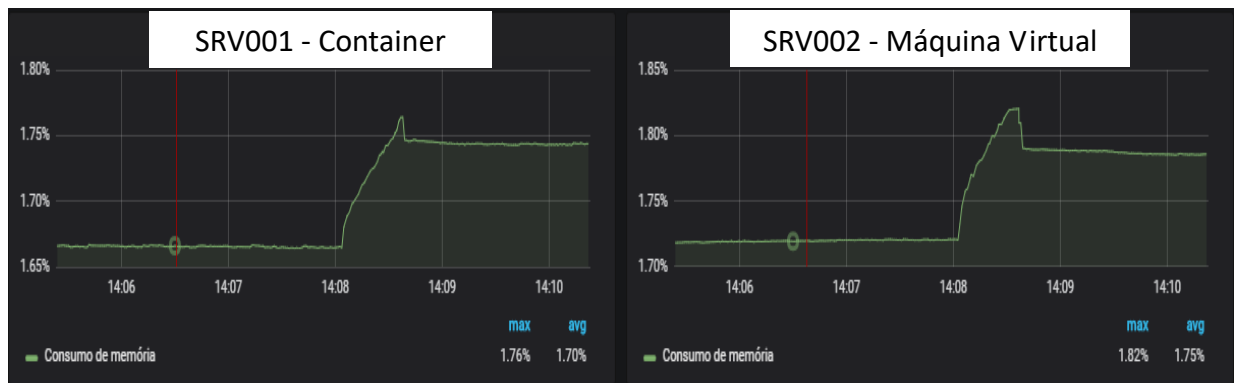


The image shows a web interface for UNIVATES. At the top is the UNIVATES logo. Below it is a section titled "Configuração". Inside this section, there are three input fields: "Tipo:" with a dropdown menu showing "APACHE AB", "Usuários:" with the value "1125", and "Páginas:" with the value "800". To the right of these fields are four icons: a lightning bolt, a lightning bolt with a globe, a lightning bolt with a document, and an eye icon.

Fonte: Do autor (2018).

Conforme demonstrado na figura 55, o teste foi executado com 800 usuários e cada usuário efetuou 1125 requisições em ambos os servidores de testes. O gráfico 9 demonstra o consumo de memória utilizada durante o teste nos servidores SRV001 e SRV002.

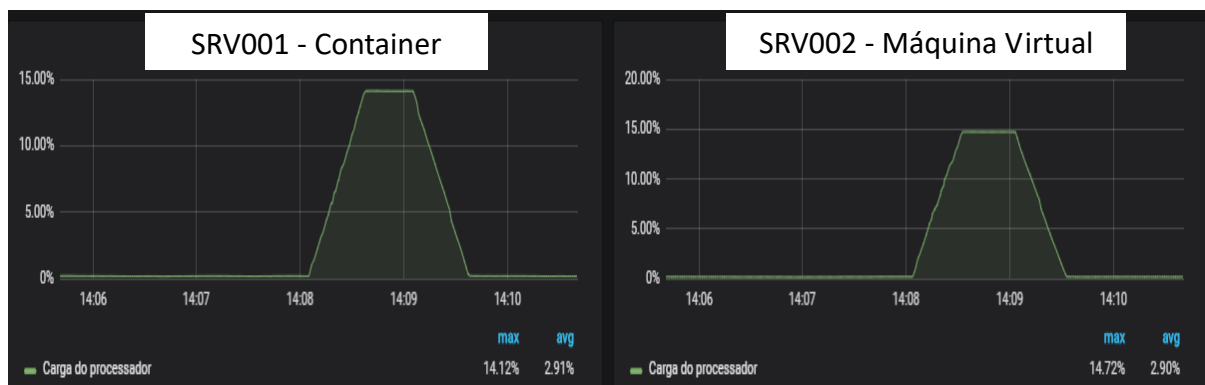
Gráfico 9 – Consumo de memória no teste com carga do Apache.



Fonte: Do autor (2018).

Conforme demonstrado no gráfico 9 os servidores de testes se comportam de uma maneira muito similar quando se trata de consumo da memória, pois, a diferença entre eles na média entre 5 minutos é de 0,05%. Já o gráfico 9 apresenta o consumo de CPU dos servidores durante o teste.

Gráfico 10 – Consumo de CPU no teste com carga do Apache.



Fonte: Do autor (2018).

De acordo com o gráfico 10 pode se analisar que o consumo de CPU entre os servidores também tem um comportamento similar neste teste, pois a diferença na média entre 5 minutos de coleta é de 0,01%. Dando sequência o gráfico 11 apresenta a leitura e a escrita do disco rígido durante o teste em questão.

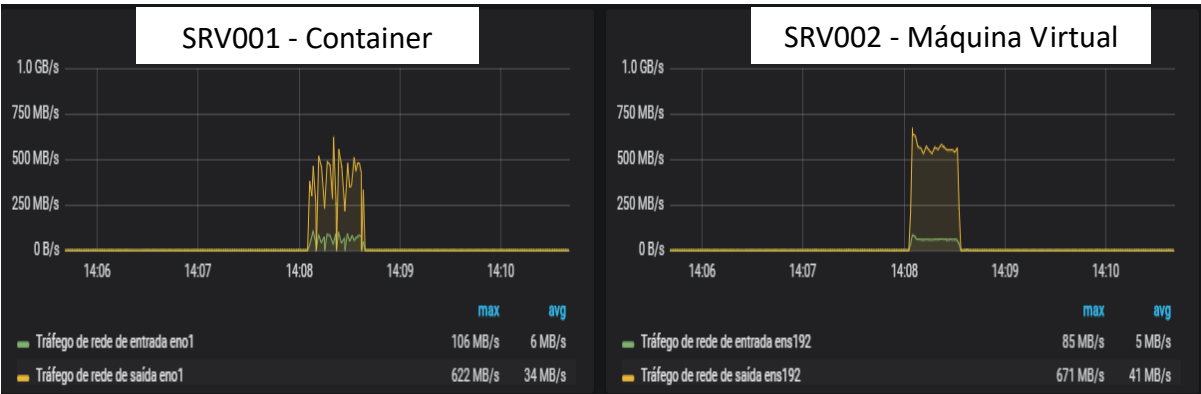
Gráfico 11 – Leitura e escrita de disco rígido no teste com carga do Apache.



Fonte: Do autor (2018).

Conforme demonstrado no gráfico 11 a leitura e a escrita do disco tem comportamento e similar, porém a taxa de escrita tem uma diferença na média de 26,08 MB/S. Lembrando que está sendo suposto que os discos de ambos os servidores estão com a mesma taxa de rotação. Para finalizar o gráfico 12 exibe o resultado do *throughput* de rede durante o teste.

Gráfico 12 – Entrada e saída de rede no teste com carga do Apache.



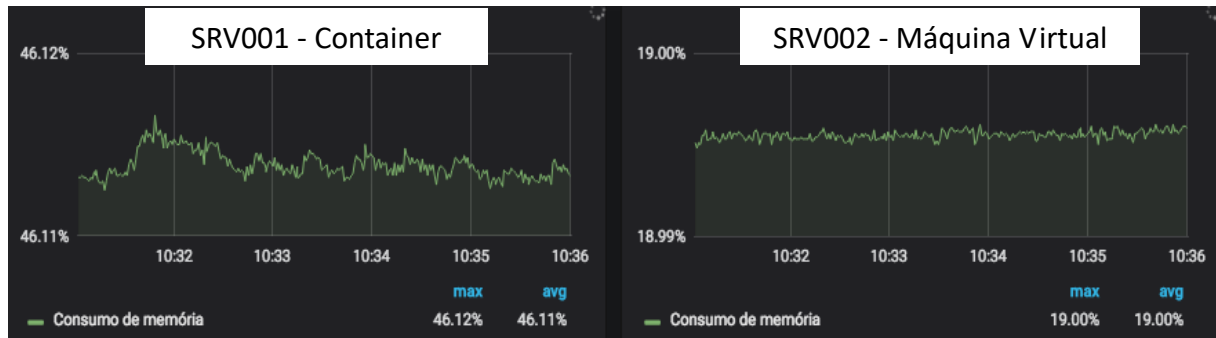
Fonte: Do autor (2018).

Conforme demonstrado no gráfico 12 o *throughput* de rede se comporta de uma diferente, pois o servidor SRV002 tem uma estabilidade maior tanto na entrada quanto na saída de dados, porém mesmo com uma variação na estabilidade a diferença na média durante os 5 minutos foi baixa. Esta diferença foi de 1MB/S na entrada e de 67MB/S na saída de dados.

6.4 Observações gerais sobre os testes

Após executado os testes descritos acima, foi colocado as maquinas em um estado ocioso durante algumas horas com o intuito de verificar como as tecnologias se comportavam (GRÁFICO 13).

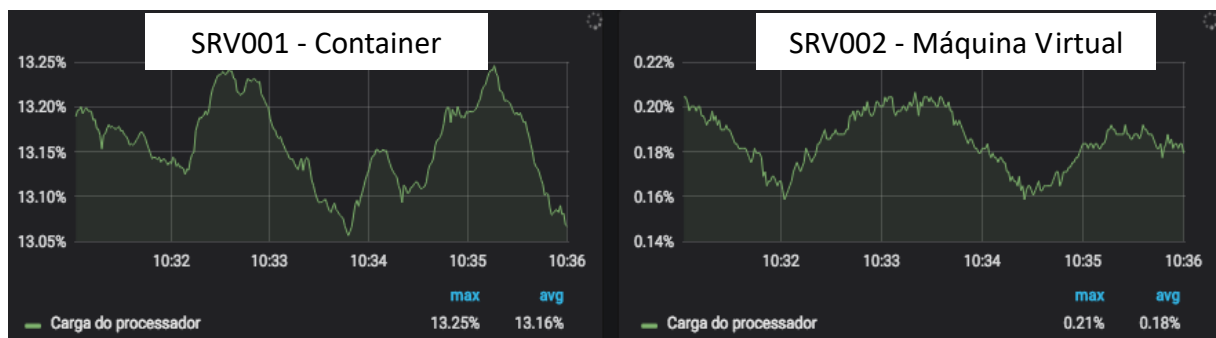
Gráfico 13 – Consumo de memória após execução dos testes.



Fonte: Do autor (2018)

Conforme demonstrado no gráfico 13 o comportamento das duas tecnologias é diferente após a execução dos testes, pois o consumo de memória tem uma diferença de 27% na média, mesmo após deixar os servidores ociosos durante algumas horas. Outra diferença entre as duas tecnologias após a execução dos testes será apresentada no gráfico 14.

Gráfico 14 – Consumo de CPU após execução dos testes.



Fonte: Do autor (2018).

Conforme o gráfico 14, também pode-se verificar que existe uma diferença grande no quesito consumo de CPU, pois deixando as maquinas de testes ociosas durante algumas horas se encontrou uma diferença na média de 12,98%.

7 CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma análise de desempenho entre containers e máquina virtuais do tipo 1, com o objetivo de comparar o consumo de memória, consumo de CPU, I/O do disco rígido e *throughput* de rede de ambas tecnologias.

A análise foi efetuada a partir da utilização em grande escala dos serviços de banco de dados e páginas web. Para conseguir efetuar tal análise sobre o banco de dados foi desenvolvido um algoritmo específico para realizar a função de *benchmark*. Quanto ao serviço de páginas web se utilizou uma ferramenta específica de *benchmark* disponibilizada pela organização Apache.

A análise do consumo de memória no teste de serviço de banco de dados se comportou de maneira totalmente diferente entre as tecnologias, pois o consumo de memória da tecnologia containerização foi superior, isso aponta que a máquina virtual do tipo 1 gerencia a memória de uma forma mais eficiente. Quanto a análise do consumo de memória no teste de página web obteve-se resultados semelhantes entre ambas tecnologias.

Quanto a análise do consumo de CPU no teste de banco de dados e no teste de páginas web se obteve um consumo semelhante ao comparar as duas tecnologias.

Já o monitoramento do I/O de disco durante os testes com serviços de banco de dados apontou variações entre as duas tecnologias no quesito gravação em disco, pois a máquina virtual do tipo 1 atingiu uma velocidade de gravação superior a tecnologia de containerização. Este resultado provavelmente está relacionado a

forma de gerenciado dos recursos físicos do hardware a partir do *bare metal*. Quanto ao I/O de disco durante os testes de página web demonstrou-se semelhante em ambas tecnologias. Este resultado pode ser caracterizado pelo fato de os serviços web não tem características que consumir recursos de I/O.

Quanto ao *throughput* de rede a máquina virtual do tipo 1 se mostrou ser superior em ambos os testes efetuados, pois, a análise efetuada demonstra uma estabilidade maior quando comparado com a tecnologia de containerização.

Os resultados obtidos através da análise dos objetivos descritos no atual trabalho, apontam que a tecnologia de máquina virtual do tipo 1 é superior em aspectos que demandam aplicações de alto desempenho, principalmente no consumo de memória RAM e I/O de disco rígido.

Por fim, é possível destacar, que os objetivos propostos para o presente trabalho foram atingidos.

Sugere-se para trabalhos futuros, considera-se importante comparar o desempenho das duas tecnologias utilizando outras ferramentas de *benchmark*, já reconhecidas para este fim. Sugere-se ainda testar o desempenho das duas tecnologias com outros serviços que explorem de modo diferenciado os recursos computacionais de memória, CPU, I/O do disco rígido e *throughput* de rede.

REFERÊNCIAS

BARROS, José R. R. **Estudo de Caso: Virtualização de servidores em ambiente corporativo**. 2016. 92f. Monografia (Graduação). Bacharelado em Ciências da Computação, Faculdade Lourenço Filho, Fortaleza (CE), 2016. Disponível em: <<http://br.monografias.com/trabalhos-pdf/estudo-caso-virtualizacao-servidores-corporativos/estudo-caso-virtualizacao-servidores-corporativos.pdf>>. Acesso em: 10 out. 2017.

CHEMIN, Beatriz F. **Manual da Univates para trabalhos acadêmicos: Planejamento, elaboração e apresentação**. 3. ed. Lajeado: Univates. 2015. 315 p.

CORRÊA, José N. G. F. Lifter - **Disponibilização de aplicações via containers de software em um cluster de alto desempenho**. 2016. 93f. Monografia (Graduação). Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis (SC), 2016. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/171410/TCC_Lifter_JoseNorberto.pdf?sequence=1&isAllowed=y>. Acesso em: 10 out. 2017.

FELTER, et al. **An updated performance comparison of virtual machines and linux containers**: Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On. IEEE, 2015.

GIL, Antônio C. **Como elaborar projetos de pesquisa**. 5. ed. São Paulo: Atlas, 2010. 184 p.

GOLDBERG, Robert P. **Princípios arquitetônicos para Virtual Computer Systems**. Ph.D. tese, 89 Universidade de Harvard, Cambridge, MA, de 1972.

LAUREANO, Marcos. **Máquinas Virtuais e Emuladores**: Conceitos, Técnicas e Aplicações. São Paulo: Novatec, 2006.

LAUREANO, Marcos Aurelio Pchek; MAZIERO, Carlos Alberto. **Virtualização**: Conceitos e aplicações em segurança. Livro-Texto de Minicursos SBSeg, p. 1-50, 2008.

IBM – INTERNATIONAL BUSINESS MACHINES, **Emulação do sistema com QEMU**. 2007 – Disponível em: <<http://www.ibm.com/developerworks/br/library/l-QEMU/>>. Acessado em: 15 de outubro 2017.

IBM **Spectrum Virtualize**. 2015. Disponível em: <<http://www.ibm.com/systems/storage/spectrum/virtualize/>> Acesso em: 24 de outubro 2017.

INTEL. **Intel VT**. Disponível em: <<http://www.intel.com/portugues/business/technologies/virtualization.htm> >. Acessado em: 08 de outubro 2017.

JUNIOR, Celso F. **Guia do trabalho científico**: do projeto à redação final. 1 ed. São Paulo: Contexto, 2011.

MACHADO, Francis Berenger. **Arquitetura de sistemas operacionais** – 1997. Disponível em: <<http://www.truenet.com.br/vivianef/SOI.html>>. Acessado em: 29 de maio 2015.

MARCONI, Marina. D. A.; LAKATOS, Eva. M. **Técnicas de pesquisa**: planejamento e execução de pesquisas, amostragens e técnicas de pesquisas, elaboração, análise e interpretação de dados. 7.ed. São Paulo: Atlas, 2008. 296p.

MATTOS, Diogo Menezes Ferrazani. **Virtualização**: VMWare e Xen. UFRJ – CENTRO DE TECNOLOGIA – DEL – 2008. Disponível em: <http://www.gta.ufrj.br/grad/08_1/virtual/VantagenseDesvantagens.html>. Acessado em: 07 de outubro 2017.

SHAUL, Ben (2011). **Hybrid Desktop Virtualization**: A New Approach for the Cloud. Virtual-strategy Magazine. Disponível em: <<http://www.virtual-strategy.com/2011/05/11/hybrid-desktopvirtualization-new-approach-cloud>>. Acessado em: 25 de outubro 2017.

ROCHA, Victor. **Tipos de virtualização**. Disponível em: <<https://www.tiespecialistas.com.br/2013/03/tipos-de-virtualizacao> >. Acesso em: 30 de outubro 2017.

SILVA, Kleider A de A. **Implementação de uma infraestrutura de cluster virtualizada com alta disponibilidade**. 2012. 66f. Monografia (Graduação). Curso Superior de Tecnologias de Redes da Computadores, Faculdade de Tecnologia de Lins Prof. Antônio Seabra, Lins (SP), 2012. Disponível em: <<http://www.fateclins.edu.br/site/trabalhoGraduacao/1I2PyX3E1QTbDohq6E9NAseI CDwDTUZCCgqg4Cxbs.pdf>>. Acesso em: 10 out. 2017.

SMITH, Jim; NAIR, Ravi. **Virtual machines**: versatile platforms for systems and processes. Elsevier, 2005.

VM WARE. **Métodos para instalar o ESXi 6.0**. Disponível em: <https://kb.vmware.com/s/article/2130503?lang=pt_PT >. Acessado em: 08 de setembro 2018.

PIRES, Aécio. **Instalação do Zabbix Server**. Disponível em: <http://blog.aeciopires.com/zabbix_4-0/ >. Acessado em: 05 de outubro 2018.

PRETEL, Jorge. **Grafana no Zabbix 3-0**. Disponível em: <<https://jorgepretel.com.br/2016/06/plugin-grafana-no-zabbix-3-0/> >. Acessado em: 08 de outubro 2018.

TWISTLOCK. **Container Whitepaper**. Disponível em: <<https://www.twistlock.com/container-whitepaper-chapter-1> >. Acesso em: 08 de outubro 2017.

VERAS, Manoel. **Datacenter componente central da infraestrutura de TI**. Rio de Janeiro: Brasport (2009).

VERAS, Manoel. **Virtualização componente central do datacenter**. Rio de Janeiro: BRASPORT, 2011.

VERAS, Manoel. **Virtualização tecnologia central do datacenter**. Rio de Janeiro: BRASPORT, 2016.

YU, Yang. **Os-level virtualization and its applications**. State University of New York at Stony Brook, 2007.